VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

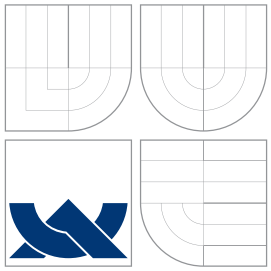
JEDNODUCHÝ SPRÁVCE OKEN PRO X WINDOW SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

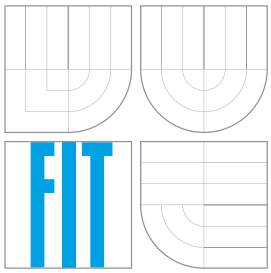
AUTOR PRÁCE
AUTHOR

JIŘÍ ZAJDÁK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

JEDNODUCHÝ SPRÁVCE OKEN PRO X WINDOW SYSTEM

SIMPLE WINDOW MANAGER FOR X WINDOW SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ ZAJDÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Dr. Ing. PETR PERINGER

BRNO 2008

Abstrakt

Cílem projektu je vytvoření jednoduchého správce oken pro X Window System. Nejprve jsou v textu vysvětlené principy výstavby grafických aplikací, dále návrh správce oken a jeho implementace. Správce oken dekoruje top-level okna aplikací rámečky, které obsahují funkční tlačítka. Pro snadnější ovládání jsou podporované virtuální plochy a klávesové zkratky. Ve spodní části obrazovky je umístěný panel určený pro zobrazení grafiky pluginů, které jsou implementované formou sdílených knihoven. Aplikace klade důraz na minimální paměťovou náročnost.

Klíčová slova

správce oken, Xlib, X okenní systém, X server, ICCCM, smyčka událostí, kurzor, keysym, modifikační klávesy, root okno, top-level okno, reparenting, atom, vstupní metoda, plugin, layout, substructure redirection, grafický kontext, keyboard focus, save-set

Abstract

Goal of the project is the creation of the simple window manager for X Window System. At first there are explained the principles of work of the graphic applications then the design of window manager and its implementation in the text. The window manager decorates the top-level windows of applications frames which contain functional buttons. For an easily operating are supported virtual screens and hot keys. The panel intended for graphic of plugins, which are implemented as shared libraries, is set in the bottom part of the screen The application place emphasis on minimal memory requirement.

Keywords

window manager, Xlib, X Window System, X server, ICCCM, event loop, cursor, keysym, modifier keys, root window, top-level window, reparenting, atom, input metod, plugin, layout, substructure redirection, graphics context, keyboard focus, save-set

Citace

Jiří Zajdák: Jednoduchý správce oken pro X Window System, bakalářská práce, Brno, FIT VUT v Brně, 2008

Jednoduchý správce oken pro X Window System

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Zajdák
6. května 2008

Obsah

Kapitola 1

Úvod

Tato práce popisuje principy, na kterých pracuje správce oken a jeho návrh. Čtenář se nejprve seznámí v kapitole (2) s problematikou grafických aplikací.

V kapitle (2.1) se vysvětluje, k čemu slouží grafický kontext, jak se nastavují jeho členy a co je třeba pro jeho vytvoření a zrušení. Další část (2.2) popisuje postup při kreslení grafických primitiv a textů. Je zde rozebrána také problematika fontů. Kapitola (2.3) ukazuje cestu, jak je možné pomocí logické funkce `XOR` vykreslovat obrys okna při jeho posouvání nebo změně jeho velikosti. Dále (2.4) jsou rozebrány jednotlivé atributy oken. Je zde vysvětlené, jak se atributy dají zapisovat, modifikovat nebo číst a které funkce se při práci s nimi uplatní. Další část (2.5) vysvětluje, jak se dají získat informace, které většinou souvisejí se zobrazením okna na obrazovce. K čemu slouží události a jak se s nimi pracuje se čtenář dozví z kapitoly (2.6). Je zde také rozebrán způsob vybírání příjmu událostí určitých typů a datové struktury s nimi související. Je zde vysvětlené, jak sestavit smyčku událostí. V kapitole (2.7) jsou vyjmenované termíny, které by měl každý správce oken podporovat. Kapitola (2.8) se zabývá kontrolou nad změnou konfigurace top-level oken aplikací a co je k tomu třeba nastavit. V (2.9) jsou popsané principy, pomocí kterých někteří správci oken orámovávají top-level okna aplikací. Kapitola (2.10) se věnuje zanořování oken pod sebou a virtuálními plochami. Dále (2.11) se čtenář dozví, jak vytvořit a vybarvit kurzor myši. Kapitola (2.12) ukazuje způsob, jak aplikace zajišťuje zpracovávání klávesových zkratek. V kapitolách (2.13 a 2.14) jsou rozebrány typy událostí, které s těmito zařízeními souvisejí a jejich další zpracování. Meziklientské komunikaci se věnuje kapitola (2.15). Je zde popsané, k čemu meziklientská komunikace slouží a jakých prostředků lze použít k její realizaci. K čemu slouží seznam `save-set` a jak se s ním pracuje, se lze dozvědět v části (2.16). Kapitola (2.17) popisuje, jak zpracovávat chyby generované knihovnou Xlib, aby nedošlo k zastavení běhu správce oken.

Následující kapitola 3 se analýzou existujících správců oken a to zejména v části 3.1. V daší části textu 3.2 jsou navrženy možná řešení pro různé části implemetace správce oken. Jsou zde také graficky zobrazeny závisloti jednotlivých modulů.

V následující části textu (4) se čtenář seznámí s konkrétní implementací správce oken. Text je členěn do kapitol podle určení jednotlivých modulů. Připojení k X serveru, inicializace základních datových struktur a hlavní smyčka událostí je popsána v kapitole (4.1). V další části textu (4.2) je vysvětlena implementace základních funkcí správce oken. Kapitola (4.3) ukazuje, jakým způsobem jsou zpracovávané události generované X serverem. Dále (4.4) se čtenář dozví, jak jsou ukládáni klienti a jaké operace se s nimi provádí. Jak probíhá vykreslování grafiky je popsané v kapitole (4.5). Další část textu (4.6) ukazuje způsob fungování pluginů zaváděných do aplikace. V kapitole (4.7) jsou rozebrané imple-

mentované pluginy. V další části textu (4.8) jsou uvedeny metriky kódu a výsledky ověřování funkčnosti na některých aplikacích.

V závěrečné kapitole (5) je shodnocené konečné řešení správce oken a možné návrhy pro pokračování projektu. Dále jsou zde popsány záludnosti nízkoúrovňového vývoje grafických aplikací.

Kapitola 2

Úvod do X Window Systému

V této kapitole budou vysvětlené základní principy, na kterých jsou postavené aplikace běžící pod X Window Systémem, zvláště správci oken. Problematika návrhu aplikací běžících pod X Window Systémem je rozebrána v knize Xlib Programmning Manual [1]. V elektronické formě je dostupná i na webových stránkách [6]. V jednotlivých částech textu se čtenář dozví, jakým způsobem grafické alikace a správci oken pracují, co mají společné, jaké funkce a datové struktury používají a jakým způsobem navzájem komunikují.

2.1 Grafický kontext

Grafický kontext definuje, jakým způsobem se budou kreslit grafická primitiva jako jsou body, čáry, texty, obrázky a další, zatímco grafická primitiva určují, co se bude kreslit. Grafické kontexty jsou ukládány na X serveru, aby se omezil síťový provoz mezi Xlib a X serverem při každém grafickém požadavku. Z toho plyne, že na různá grafická primitiva můžeme použít jedinou hodnotu grafického kontextu.

Při práci s grafickými kontexty můžeme buďto měnit hodnoty jednoho grafického kontextu nebo použít více grafických kontextů s různými hodnotami a vybírat podle potřeby mezi samotnými grafickými kontexty. Používání grafického kontextu v programování má zejména výhodu v malém počtu předávaných argumentů grafickým funkcím.

Grafický kontext se vytváří voláním funkce `XCreateGC`. I přesto, že je jeden z jejích parametrů `drawable`, nemusí být grafický kontext použit ke kreslení jen do jednoho okna. Grafický kontext se může použít ke kreslení do jakéhokoliv okna. Důležité ovšem je, aby zůstala ta samá barevná hloubka a ten samý `screen` jako u daného `drawable`. Proměnné grafického kontextu se nastavují ve struktuře `XGCValues`. `Valuesmask` potom určí, které proměnné chceme nastavit.

2.2 Kreslení primitiv a vypisování textů

knihovna Xlib nám pro kreslení grafiky nebo textů nabízí mnoho různých specializovaných funkcí od kreslení bodů, čar, až po složitější primitiva jako jsou obdélníky či kruhy nebo elipsy. S Xlib můžeme také vypisovat texty a podporované jsou i černobílé bitmapy.

Kreslení grafických primitiv

Předtím, než začneme kreslit, musíme přinejmenším nastavit dvě proměnné grafického kontextu a to jsou hodnoty pixelů popředí a pozadí. Pro černobílé aplikace je možné pro

nastavení těchto proměnných použít makra `BlackPixel` a `WhitePixel`. Pro získání jiných barev je nutné volat funkce pro alokaci barev, např. funkci `XAllocNamedColor`. Pro zrušení barvy potom `XFreeColor`. U uzavřených primitiv, jako jsou např. obdélníky nebo kruhy, máme možnost kreslit buď jen obrys nebo použít fukce, které mají v názvu "fill" vykreslující primitiva vyplněná. Proces kreslení obvykle spočívá v následujících krocích:

- Nastavení potřebných proměnných grafického kontexu.
- Zjištění rozměrů cíle, do kterého chceme kreslit.
- Kreslení samotné.

Je třeba počítat s tím, že parametry fukcí pro kreslení vyplněných primitiv mohou být odlišně interpretované než parametry pro kreslení obrysu primitiva. Např. šířka a výška vyplněného obdélníku je o jeden pixel menší než při kreslení jeho obrysu při stejných parametrech kreslících funkcí.

Vypisování textů

Fonty jsou v Xlib utvořené z bitmap. Mohou reprezentovat text, tvary kurzorů nebo jiné tvary pro speciání použití. Fukce pro manipulaci s fonty mají dvě varianty. Jedna varianta pracuje s 8-bitovými fonty a druhá varianta s 16-bitovými fonty. Rozdíl mezi nimi je v rozsahu znaků, které lze použit. 8-bitové fonty umožňují zobrazit 256 znamků, zatímco 16-bitové umožňují zobrazit 65536 znaků. 16-bitové fonty se proto používají pro neanglické jazyky.

Více klientů může sdílet na serveru jednu kopii fontu. Dřive než ji začnou používat, je nutné, aby byl font alokován funkcí `XLoadFont`. Dostupné fonty na serveru je možné zjistit voláním funkcí `XListFonts` nebo `XListsFonsWithInfo`, které vracení pole s názvy fontů. O zrušení takto získaného pole se postará funkce `XFreeFontNames`. Některé fonty jako např. "fixed" nebo "9x15" by měly být dostupné vždy. Při použití takových fontů je volání funkce `XListsFonts` zbytečné. Pokud máme vybraný název fontu, alokujeme font pomocí funce `XLoadFont`. Funkce vrací ID fontu, kterým se na font odkazuje. Funkcí `XSetFont` nakonec asocicuje font s daným grafickým kontextem.

Atributy fontu jsou uložené ve struktuře `XFontStruct`. Tuto strukturu získáme voláním funkce `XQueryFont`. K dealokování fontu slouží `XFreeFont` nebo `XUnloadFont`. Často je potřeba znát šířku, někdy i výšku řetězce v pixelech, který vypisujeme. Problém nastává u fontů s proměnnou délkou znaků, kdy např. písmeno "i" je kratší než písmeno "m". S tímto problémem pomohou funkce `XTextWidth` a `XTextHeight`. Funkce berou řetězec a příslušnou strukturu `XFontStruct` a posléze vrátí rozměry v pixelech. Funkce `XDrawString` potom slouží k samotnému vypsání textu na danou pozici.

2.3 Kreslení obrysu okna při přesunu nebo změně velikosti

Správci oken, kteří se snaží snížit spotřebu systémových zdrojů, nevykreslují při posunu nebo změně velikosti top-level okna jeho obsah, ale jen jeho obrys. Obsah okna se překreslí až ve chvíli, kdy je znám jeho nový konečný stav.

Při vykreslování obrysu okna se využívá logické operace `XOR`. Při použití logické operace `XOR` nám odpadá povinnost si pamatovat hodnoty překreslených pixelů. Z principu operace

XOR vyplývá, že pokud je obrys okna překreslen znovu stejným obrysem, vrátí se hodnoty pixelů do původního stavu a tím obrys okna zmizí.

Před započetím operací, které vytvářejí obrys okna, je ovšem nutné volat funkci XGrabServer, aby se zamezilo zpracovávání požadavků ostatních klientů. Kdyby se tomu nezamezilo a některý zúčastněný klient se např. rozhodl překreslit obsah okna, vznikly by ve spojení s operací XOR v tomto klientu různobarevné čáry, které by se odstranily až s dalším požadavkem na překreslení. Po ukončení vykreslení obrysu okna se zavolá funkce XUngrabServer, která opět umožní zpracovávání požadavků ostatních klientů. Ukázka aplikace logické operace XOR pro jeden pixel obrysu:

```
barva pozadí              1111 0000 1111 0000 1111 0000
operace XOR
nastavená barva obrysu    1010 1010 1010 1010 1010 1010
                          ----------------------------
skutečná barva obrysu     0101 1010 0101 1010 0101 1010
operace XOR
nastavená barva obrysu    1010 1010 1010 1010 1010 1010
                          ----------------------------
barva pozadí              1111 0000 1111 0000 1111 0000
```

2.4 Atributy okna

Atributy okna definují základní vlastnosti okna. Pro jejich nastavení lze použít funkci XSetWindowAttributes. Pro jejich modifikaci funkci XChangeWindowAttributes. Získání atributů okna se potom dosáhne voláním funkce XGetWindowAttributes. Je důležité si uvědomit, že s těmito operacemi souvisí dvě různé struktury. Jedna je pro nastavení atributů - struktura XSetWindowAttributes a druhá pro jejich čtení - struktura XWindowAttributes. Atributy okna můžeme měnit buď po skupinách, kdy využijeme bitové masky nebo jednotlivě použitím specializovaných funkcí.

Mezi atributy, které ovlivňují vzhled okna, můžeme zařadit background_pixel, background_pixmap, border_pixel, border_pixmap, colormap a cursor. Klienti často nastavují ohraničení okna a kurzor. Colormapu obvykle nechávájí výchozí. Ohraničení oken správci oken obvykle odstraní. Další atributy nastavují chování okna při jeho překreslování nebo při změně velikosti. Sem můžeme zařadit backing_pixel, backing_planes, backing_store, který definuje, kdy a za jakých okolností je uchován obsah okna na serveru. Bit_gravity určuje, na jaké pozici se ocitne původní část okna při změně velikosti. Save_under definuje způsob uchování obsahu překryté části okna. Atributem event_mask vybereme příjem požadovaných události. Do_not_propagate_mask potom zajistí, které události se budou propagovat oknu předchůdci. Win_gravity atribut definuje, jak se budou přeskupovat okna následníci při změně velikosti okna. Poslední atribut override_redirect značí, že správce oken nemá spravovat toto okno. Správce oken nebude mít možnost zachytit požadavky na mapování nebo změnu konfigurace okna.

2.5 Layout politika

Správce oken může získat informace o layoutu okna pomocí funkcí XGetNormalHints nebo XGetWMHints. Funkce vrací ukazatel na strukturu XSizeHints. Členy x, y, width, height jsou zastaralé a jsou zde jen kvůli zpětné kompatibilitě. K jejich získání je lépe použít

např. funkci `XGetWindowAttributes`. Členy `min_width` a `min_height` určují minimální použitelné rozměry okna, `max_width` a `max_height` potom maximální povolené rozměry okna. Členy `width_inc` a `height_inc` určují, po kolika pixelech je možné okno zvětšovat nabo zmenšovat v každém směru. `Base_width` a `base_height` určují požadované rozměry okna. Správce oken může měnit jen layout top-level oken. Nemá žádné prostředky pro změnu layoutu jejich následníků.

2.6 Události

Událostmi řízené programování je značně odlišné od programování standardního. Pro toto programování je typické, že program reaguje na aktuální akce uživatele. Programy si navzájem konkurují a musí sdílet jedno grafické prostředí. To znamená, že se mohou navzájem ovlivňovat. Události plní zpravidla následující úlohy:

- Dávají informace o všem, co program potřebuje znát k jeho běhu. Např. informace o uživatelském vstupu.
- Podávají informace jiným klientům o tom, co program dělá.
- Informují správce oken, pokud např. klient žádá o překonfigurování okna.

X server generuje události a ukládá je do fronty. Události jsou rozesílány klientům, kteří o ně požádali. Klientská aplikace si vybere funkcí XSelectInput typy událostí, které má zájem přijímat. Pokud mají klienti vybraný stejný typ události, potom X server vytvoří kopii takové události a rozešle ji všem zájemcům.

Pro úsporu síťového provozu je dobré, když si klient nevybere typy událostí, na které nechce reagovat. Po příchodu události může klientská aplikace vyvolat akci reagující na danou událost. Je-li ve frontě další událost, postup se opakuje. Program obvykle nemá možnost předpovídat, jaká událost se vyskytne v budoucnu.

Výběr typů událostí

Postup obsluhy událostí pro klientskou aplikaci je následující:

- Klient voláním funkce `XSelectInput` vybere pro konkrétní okno události, které chce přijímat.
- Voláním funkce `XMapWindow` namapuje dané okno.
- Klient vytvoří smyčku událostí, ve které načítá příchozí události zvolených typů a vyvolává příslušné akce.

Toto pořadí je dobré striktně dodržovat. Pokud by klient začal provádět smyčku událostí dříve než by nastavil voláním funkce `XSelectInput` typy událostí, které chce přijímat, mohlo by dojít k situaci, kdyby přišel např. o první událost typu `Expose` a nemohl by vykreslit obsah okna. Pokud by se jednalo o top-level okno, mohl by klient také přijít o událost typu `ConfigureNotify`, která by se vygenerovala např. po vykonání změny velikosti okna správcem oken. Z výše uvedeného plyne, že jedna aplikace mající více oken, může pro každé okno nastavit příjem různých typů událostí.

```
unsigned long event_mask = ExposureMask | ButtonPressMask | KeyPressMask;
XSelectInput(display, window, event_mask);
```

Po vytvoření okna je možné měnit typy přijímaných událostí voláním funkce `XChangeWindowAttributes` a nastavením příslušných bitů masky `event_mask`.

Datové typy

Základním prvkem je union `XEvent`. Prvním členem unionu je typ události. Ten se vyskytuje ve všech případech. Union dále sdružuje struktury pro všechny typy událostí. Jakmile klient načte z unionu `XEvent` typ události, ví, kterou strukturu union `XEvent` obsahuje a může přistupovat k jeho datům.

Začátek všech typů struktur je stejný. Obsahuje informace o typu události, číslo požadavku jenž událost vygeneroval, příznak značící zda událost vygeneroval server nebo zda byla poslána voláním funkce `XSendEvent`, display a okno, kde událost nastala. Další členy struktury jsou specifické pro různé typy událostí.

Návrh smyčky událostí

Smyčky událostí mají zpravidla podobnou strukturu. Jsou tvořené jedním nekonečným cyklem, ve kterém se nachází funkce `XNextEvent`. Ta zkopíruje první událost z fronty klientské aplikace do struktury, kterou má v parametru. Posléze smaže tuto událost z fronty klientské aplikace. Jestliže se ve frontě klientské aplikace žádná událost nenachází, funkce čeká, dokud nepřijde další událost. Za funkcí `XNextEvent` následuje zpravidla příkaz switch, který rozřadí příchozí události podle jejich typů.

```
while(true) {
      ...
      XNextEvent(display, &event);
      while (XPending(display)) {
           switch(event.type) {
                  case expose:
                           ...
                           break;
                  case buttonpress:
                           ...
                           break;
                  ...
           }
      }
}
```

Propagování událostí

Jedním členem, kterým disponuje struktura události jakéhokoliv typu, je odkaz na okno, ve kterém k události došlo. Pro události typů `ButtonPress`, `ButtonRelease`, `KeyPress`, `KeyRelease` a `MotionNotify` to ovšem nemusí platit. Které okno je uváděno jako zdroj události ve struktuře těchto událostí, může záviset na výsledku propagování skrz hierarchii

oken. Propagování závisí na nastavení `event_mask` a `do_not_propagate_mask` struktury `XWindowAttributes`.

Jestliže vznikla událost v okně, které není nastaveno pro příjem takového typu události, událost se propaguje v hierarchii oken směrem k předchůdcům. Pokud však okno nejbližšího předchůdce má nastavenou `event_mask` pro příjem události daného typu, propagování události u tohoto předchůdce skončí. `Nenachází-li` se v hierarchii oken žádné okno, které by událost zpracovalo, X server takovou událost vůbec nepošle. Někdy je vhodné zamezit propagování událostí některých typů. K tomu slouží maska `do_not_propagate_mask`. Události typů uvedených v této masce se nebudou dále propagovat.

Keyboard focus

V současné době je standardní chování, že správci oken určují okno mající keyboard focus sofistikovanějšími způsoby, než je pouhé přesunutí kurzoru myši nad určité top-level okno.

Keyboard fucus určuje propagování událostí typů `KeyPress` a `KeyRelease`. Top-level okno, které má nastavený keyboard focus, přijímá vstup z klávesnice normálně. Pokud však událost `KeyPress` či `KeyRelease` vznikla v jiném okně (kurzor se nacházel nad jiným oknem, než tím, který má keyboard focus), je tato událost přesměrována oknu s keyboard focusem. Funkce pro nastavení keyboard focusu je `XSetInputFocus`. Okno, které má nastavený keyboard focus, ovšem musí být viditelné. Parametr funkce `rewert_to` určí, kerému oknu se keyboard fucus předá, pokud podmínka viditelnosti okna bude porušena.

Grabování klávesnice a myši

Pokud má klient nastaven keyboard nebo pointer grabbing, události jsou mu posílány přednostně před ostatními klienty, kteří mají vyžádaný příjem událostí daného typu.

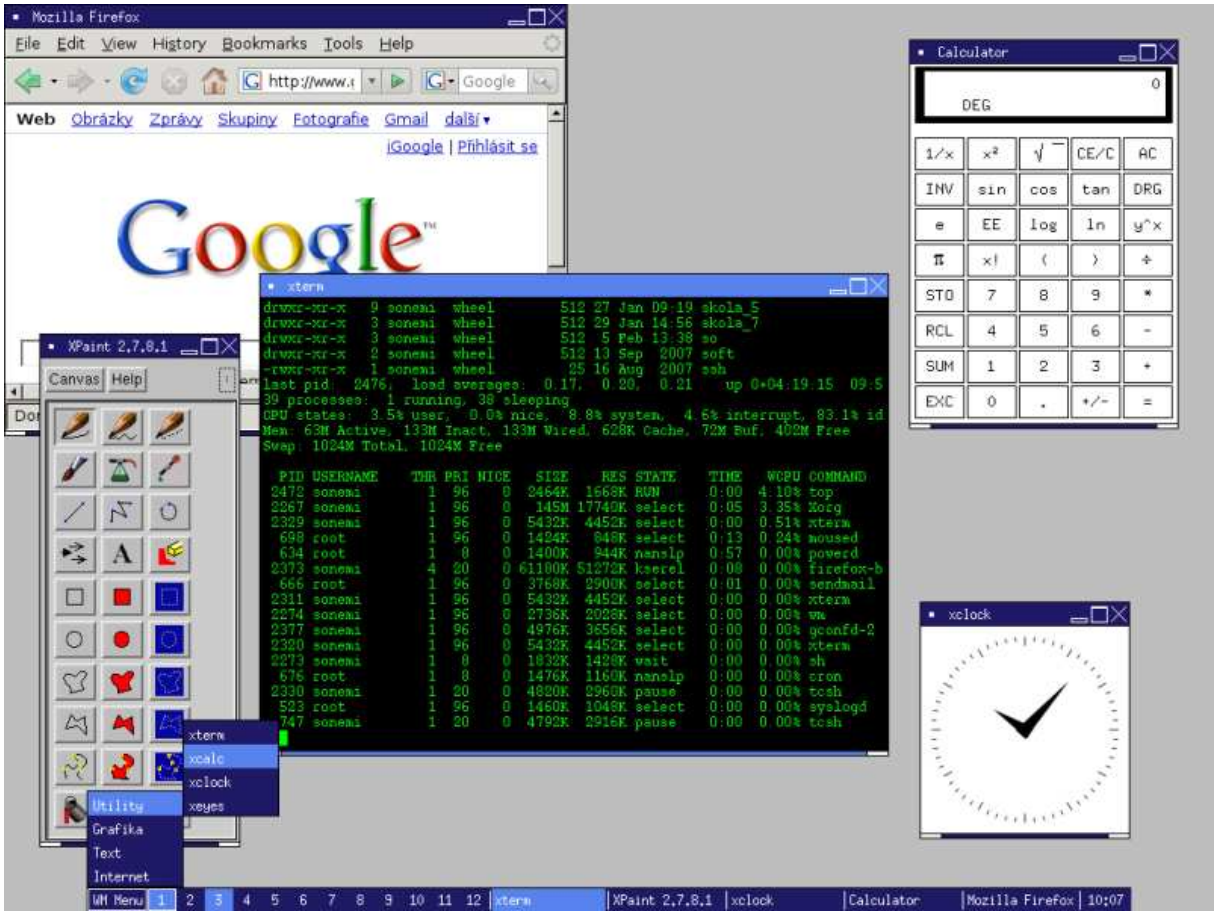
Jestliže je klávesnice nebo myš v asynchronním módu, server posílá grabované události grabujícímu klientu jakmile se vyskytnou. Pokud je klávesnice nebo myš v synchronním módu, server pozastaví vykonávání a rozešle grabované události grabujícím klientům. Vykonávání bude pokračovat až s voláním funkce `XAllowEvent` nebo s `ungrab` požadavkem.

Grabovací proces může být aktivní nebo pasivní. Aktivní grabování se nastaví voláním funkcí `XGrabPointer` nebo `XGrabKeyboard`. Pasivní grabování se nastaví voláním funkcí `XGrabKey` nebo `XGrabButton`. Pasivní grabování se spustí až se stiskem určité klávesy nebo tlačítka, případně určitou kombinací.

2.7 Hlavní vlastnosti správců oken

Správce oken je program pro správu top-level oken aplikací. Správa top-level oken aplikací zahrnuje především posouvání okna po ploše, změnu jeho rozměrů a jeho ikonyzování. Správci oken také musí být schopni spravovat zanořování oken vzájemně pod sebou.

Kromě těchto základních dovedností i jednoduší správci oken nabízejí uživateli podporu pro zjednodušení práce s grafickým prostředím. Jsou to různá menu pro rychlý výběr požadovaných činností, panely pro lepší přehled dění na pracovní ploše, případně virtuálních plochách a orámování oken obsahujících tlačítka pro nejběžnější úkony.



Obrázek 2.1: Běžící správce oken s pluginy

Následující výčet vlastností by měl podporovat každý správce oken:

- `Substructure redirection` pro zachycení požadavku na změnu layoutu okna. Je potom na správci oken, zda požadavku vyhoví zcela, částečně nebo zda ho bude zcela ignorovat.
- `Reparenting` využívající se u správců oken, které obalují top-level okna rámečky. Správce oken mění předchůdce top-level okna z root okna na okno rámečku.
- `Save-set` zabezpečující navrácení oken do jejich původního stavu, pokud je správce oken ukončen nekorektně.

2.8 Substructure redirection

Substructure redirection umožňuje správcům oken získat kontrolu nad změnou layoutu top-level oken. Správce oken nastaví masku `SubstructureRedirectMask` root okna. Tento krok způsobí, že snaha top-level okna na změnu jeho konfigurace skončí nezdarem. Místo toho se pošle správci oken událost popisující požadovanou změnu. Správce oken potom rozhodne, zda požadavek na překonfigurování top-level okna vykoná nebo zda ho vykoná

s úpravami nebo zda požadavek zamítne úplně. Správce oken tak získá kompletní kontrolu nad pozicí, rozměry, zanořením, šířkou rámečku a stavem namapování okna. Maska `SubstructureRedirectMask` umožňuje zpracování událostí následujících typů:

- `ConfigureRequest` při použití fukcí `XMoveWindow`, `XResizeWindow`, `XMoveResizewindow`, `XConfigureWindow`, `XLowerWindow`, `XRaiseWindow` nebo `XSetWindowBorderWidth`.
- `MapRequest` při použití funkcí `XMapWindow` nebo `XMapSubwindows`.
- `CirculateRequest` při použití funkcí `XCirculateSubwindows`, `XCirculateSubwindowsDown` nebo `XCirculateSubwindowsUp`.

Substructure redirection se dá použít jen na top-level okna, která nemají nastavený bit `override_redirect`. Tento bit značí, že okno si nepřeje být spravováno správcem oken. Tohoto využívají zejména pop-up okna a různá menu.

2.9 Reparenting

Správci oken často přidávají top-level oknům orámování. Orámování může obsahovat další tlačítka např. pro minimalizaci, maximalizaci nebo uzavření okna, dále tlačítka pro změnu rozměrů okna, případně tlačítka pro práci s virtuálními plochami. Obvykle bývá zobrazené i pole s nápisem názvu aplikace.

Princip spočívá ve vytvoření okna rámečku, které je následníkem root okna. Okno rámečku musí mít vhodný rozměr, aby se do něho dalo původní top-level okno patřičně umístit. Pokud to správce oken umožňuje, jsou do okna rámečku přidána další okna mající funkci tlačítek. Voláním funkce `XReparentWindow` se nakonec změní předchůdce původního top-level okna z root okna na okno rámečku.

Top-level okna s nastaveným bitem `override_redirect` nemohou být správcem oken spravované, protože nastavení tohoto bitu zamezí generování událostí s příslušnými požadavky. Správce oken může orámovávat i okna s nastavenou vlastností `X_WM_TRANSIENT_FOR`, u kterých bývá často menší počet tlačítek v rámečku okna, protože tato okna často neumožňují minimalizaci nebo maximalizaci.

2.10 Zanořování oken

Pokud správce oken chce řídit míru zanoření top-level oken na obrazovce, musí obsahovat speciální strukturu se členy odkazující na top-level okna. K této struktuře potom musí mít naimplementované vlastní funkce pro vyhledávání a práci s těmito top-level okny. Základní vlastností je možnost postupně přepínat top-level okna na obrazovce. K tomu je potřebné znát míru zanoření jednotlivých oken, aby se top-level okna přepínala podle určitého předpisu, nikoliv náhodně.

Nejčastější případ je, že okno, které je na vrcholu, se při přepnutí okna přesune dolů. Okno, které bylo bezprostředně pod ním, se tím dostane na vrchol.

K výše uvedenému postupu by se nabízela funkce `XCirculateSubwindowsDown`. Tato funkce ovšem rotuje jen s okny, které jsou navzájem překryté. To znamená, že pokud máme na obrazovce dvě skupiny oken, které jsou navzájem překryté, ale přitom se tyto dvě skupiny navzájem nepřekrývají, funkce `XCirculateSubwindowsDown` bude cirkulovat jen s jednou z těchto dvou skupin. Proto je vhodné naimplementovat funkci vlastní, která bude cirkulovat se všemi okny.

Další oblast, kde se speciální struktura a funkce pro práci s top-level okny využije, jsou virtuální plochy. Je nutné při výběru některé z virtuálních ploch nepatřičná top-level okna odmapovat a patřičná zase namapovat. Při tomto kroku by ovšem nebylo dobré, kdyby se na jednotlivých plochách změnilo pořadí zanoření oken. Jediný případ, kdy tento stav může nastat, je při připnutí některého top-level okna na všechny plochy.

V tomto případě se může stát, že toto připnuté top-level okno, které máme např. na první virtuální ploše na samém dnu, zcirkulujeme na druhé virtuální ploše na vrchol. Po přepnutí zpět na první virtuální plochu se připnuté top-level okno zobrazí na vrcholu, nikoliv na dně.

2.11 Kurzory

Ke každému oknu můžeme pomocí atributu okna přiřadit kurzor. Kurzor a jeho tvar se vytvoří voláním funkce `XCreateFontCursor`. Správce oken obvykle nastaví pro root okno kurzor tvaru šipky. Při speciálních akcích, jako je posun okna po obrazovce nebo změna velikosti okna, se zpravidla zobrazí jiné tvary kurzoru vhodně adrážející tuto situaci.

Tvary kurzorů je možné vybírat z množství předdefinovaných vzorů a velikostí. Pokud nevyhovuje standardní černé pozadí a bílé popředí kurzoru, je možné ho vybarvit voláním funkce `XRecolorCursor`.

2.12 Klávesové zkratky

Správci oken nabízí klávesové zkratky pro často opakované činnosti, jako je např. přepínání top-level oken nebo virtuálních ploch či spouštění často používaných programů. Využití této techniky uživatelům velmi šetří čas.

Při využívání klávesových zkratek je nutné zajistit, aby událost o stisku vybraných kláves získal jen správce oken a ne okno klientské aplikace ve kterém ke stisku došlo. Proto správce oken volá na root okno funkci `XGrabKey`. Tato funkce nagrabuje potřebné klávesy a případně i klávesy s potřebnými modifikátory. Modifikátory jsou klávesy (např. “Alt” nebo “Shift”) měnící význam kláves, které jsou stisknuté zároveň s nimi. Správce oken potom dostává události od takto nagrabovaných kláves a může po jejich přijetí vyvolat odpovídající rutiny.

2.13 Klávesnice

Klávesnice generuje události informující, která klávesa byla stlačena či uvolněna. Události typů `KeyPress` a `KeyRelease` jsou uloženy ve struktuře `XKeyEvent`. Důležitými členy této struktury jsou zejména `keycode` stisknuté klávesy a `state`, což je maska určující, které modifikující klávesy nebo tlačítka byly přitom stisknuty. Modifikační klávesy jsou např. Alt, Ctrl, Shift. Tyto klávesy obvykle pozměňují význam kláves stisknutými s nimi. Stisk modifikačních kláves generují události s unikátním `keycodem` stejně jako jiné klávesy.

`Keycode` je číslo, které nabývá hodnot od 8 do 256. Toto číslo je stejné jak pro událost `KeyPress`, tak i pro událost `KeyRelease`. `Keycode` pro určitou klávesu se nikdy nemění. Ovšem klávesy se shodným symbolem na klávesnicích různých výrobců mohou generovat odlišné `keycody`. Toto je důvod, proč klient nemůže určit význam klávesy z `keycodů`. Význam klávesy představuje tzv. `keysym`. `Keysym` je symbolická konstanta začínající předponou “XK_” (XK_a, XK_A, XK_Tab, XK_F1, XK_Shift_L). Pro přeložení `keycodu` na `keysym` lze

použít funkci `XKeycodeToKeysym`. Mapování fyzických kláves na `keycody` je pevně definováno X serverem. Mapování `keycodů` na `keysymy` je implementováno tabulkou a jde modifikovat aplikacemi `setxkbmap` nebo `xmodmap`.

Národní jazyky většinou potřebují zadávat mimoanglické znaky. To se řeší postupným stiskem kláves, ze kterých se složí výsledný znak. V českém jazyce je to například psaní dlouhých samohlásek, kdy je třeba nejdříve stisknout "mrtvou klávesu čárka" a následně stisknout samohlásku. Tento proces řeší tzv. vstupní metody, které převedou soustavu stisknutých kláves na jediný tisknutelný znak.

2.14 Myš

Myš je jedním z nejdůležitějších zařízení pro práci s X Window Systémem. Generuje události informující o změně pozice kurzoru, překročení hranice okna či stisku nebo uvolnění některého tlačítka. Pozice kurzoru je definována dvěmi souřadnými osami s počátkem nacházejícím se vlevo nahoře okna.

Zjišťování cesty kurzoru

První možností je zpracovávání všech událostí typu `MotionNotify`. Program musí zpracovat každou příchozí událost typu `MotionNotify`, jejichž počet bývá velmi vysoký. Největší nevýhoda tohoto řešení spočívá ve skutečnosti, že příchozí události typu `MotionNotify` se mohou značně opožďovat za skutečnou aktuální pozicí kurzoru na obrazovce. Proto tento postup není vhodný pro psaní aplikací, kde je vyžadována okamžitá korespondence s kurzorem.

Další možností je čtení z bufferu, ve kterém jsou uloženy pohybové události v určitém časovém intervalu. Funkci `XGetMotionEvents` předáme okno a časový interval, pro který chceme získat požadované události. Funkce potom vrátí všechny události v tomto intervalu z bufferu pohybových událostí.

Poslední možností, jak získat údaje o pozici kurzoru, je volání funkce `XQueryPointer`. Tento způsob zamezuje posílání velkého množství pohybových událostí, ovšem za cenu, že získáme jen aktuální pozici kurzoru. Toto je využívané zejména aplikacemi, které potřebují získat finální pozici kurzoru po ukončení pohybu.

Překračování hranice okna

Někteří správci oken mění zanoření oken při pouhém přejetí kurzoru z jednoho top-level okna to druhého. Ke zjištění, že došlo k tomuto přesunu, je správci oken zaslána jedna ze dvou typů událostí. Prvni událost je typu `LeaveNotify`, informující, že kurzor se přesunul z daného okna. Druhá událost je potom typu `EnterNotify`, která je zaslána, pokud se kurzor přesunul nad dane okno. S tímto procesem u takových správců oken úzce souvisí nastavování keyboard focusu. Správce oken nastaví keyboard focus voláním funkce `XSetIntputFocus` top-level oknu, které generovalo událost typu `EnterNotify`. Naopak top-level oknu, které bylo zdrojem události typu `LeaveNotify` je keyboard focus odejmut.

2.15 Meziklientská komunikace

Komunikace mezi klienty je nutná, aby aplikace mohly spolupracovat a mohly korektně sdílet systémové prostředky. Meziklientská komunikace musí umožňovat aplikacím také

sdílet data, protože mnohé aplikace jsou schopny data jiným aplikacím posílat i data od jiných aplikací přijímat.

Xlib podporuje tři mechanizmy pro komunikaci mezi klienty: properties, selections a cut buffers. Pravidla komunikace jsou dány manuálem nazvaným Inter-Client Communication Conventions Manual (ICCCM). Komunikace mezi klienty se odehrává pomocí properties. Jedna aplikace properties nastaví a ostatní aplikace je mohou číst. Pro implementaci správce oken jsou důležité zejména properties, protože veškerá komunikace mezi ním a aplikacemi probíhá s jejich pomocí.

Properties a atomy

Property jsou pojmenovaná a typovaná data. X system má vlastní předdefinované properties, ale je tu možnost nadefinování vlastních dat a jejich asociování s oknem. Property mají názvy typu ASCII string. Každá pojmenovaná property má přidělen atom, což je unikátní identifikátor. Každá property má typ, např. string nebo integer. Mohou být definovány i nové typy. Jeden název property může být spojen jen s jedním datovým typem. Atomy jsou z efektivních důvodů používány spíše než názvy properties. K získání atomu z názvu property slouží funkce `XInternAtom`.

Properties mohou být na serveru uloženy jako 8-bitová, 16-bitová nebo 32-bitová data. Tahle data mohou být buďto struktury nebo raw data. X server umožňuje prezentovat data v bytovém uspořádání, jaký klient vyžaduje. Dříve než jsou data poslána na server, musí být klientem zakódována do jedné ze tří bytových možností. Server si data po jejich načtení rozkóduje. Vytvořená properties zůstavají uložené na serveru dokud klient, který je vytvořil není ukončen. Klient se tedy může odkazovat na atom vytvořený druhým klientem, jen jestliže druhý klient nebyl ještě ukončen. X Window System poskytuje 68 předdefinovaných properties. Předdefinované properties začínaji předponou "XA_". Properties můžeme nastavit funkci `XChangeProperty` a číst je funkci `XGetProperty`. Po volání funkce `XChangeProperty` je generována událost typu `PropertyNotify`.

Komunikace správcem oken

Aby správce oken mohl správně pracovat, musí být ustanovená pravidla, jak se mají klientské aplikace chovat. Které činnosti musí provádět, které činnosti mohou provádět a které činnosti naopak provádět nesmí. Základní věcí je, že klientská aplikace nastaví určité vlastnosti, které mají informační charakter pro správce oken. Klientské aplikace by neměly být navrhovány pro jednoho konkrétního správce oken, ale naopak by měly být funkční s jakýmkoliv správcem oken. Výběr správce oken by měl zůstat na volbě uživatele.

Správci oken se řídí při spravování oken tzv. hinty. Hinty obvykle nastavují klientské aplikace. Podávají správci oken jakýsi návrh, jak by mělo být něco vykonáno nebo nastaveno. Správci oken by se měli těmito hinty při spravování oken řídit co nejvíce, ale není nutné je dodržovat striktně. Aplikace tedy musí být schopná korektně běžet, i když některé hinty nejsou interpretovány úplně nebo jsou ignorovány. Klientská aplikace musí nastavit WM hint properties pro každé své top-level okno, protože správci oken vyžadují informace o těchto oknech.

Standardní properties pro správce oken

Před tím, než klient své top-level okno namapuje, je nutné, aby nastavil alespoň nutné properties, které správce oken potřebuje znát. Properties se nastavují voláním funkce

`XSetProperties`. Nastavení některých properties je nutné, nastavení jiných je volitelné. Pro korektní práci správce oken by měl klient volat funkci `XSetProperties` pro každé jeho top-level okno. Tím může poskytnout správci oken následující informace:

- Název aplikace
- Název ikony
- Příkaz pro spuštění aplikace
- Ikonu s maskou nebo okno
- Upřednostňovanou pozici ikony
- Rozměrové hinty pro běžný stav
- Spouštěcí stav
- Keyboard focus model
- Okenní skupinu

Je jen na správci oken, jak se zařídí v případě, že některá z výše uvedených vlastností nastavena nebude.

2.16 Save-set

Předtím, než správce oken zavolá na top-level okno vytvořené jiným klientem, zavolá funkci `XReparentWindow` nebo `XIconifyWindow`, přidá toto okno do seznamu zvaného `save-set`. Pokud by později došlo k nekorektnímu ukončení správce oken, např. neošetřenou chybou nebo použitím příkazu `kill`, oknům ze seznamu `save-set` bude změněn přechůdce zpět na root okno.

Přidávání oken do seznamu `save-set` se děje voláním funkce `XAddToSaveSet`. Odebírání ze seznamu `save-set` je automatické a děje se tak při zrušení takového okna. Pokud je nutné explicitně změnit obsah seznamu `save-set`, lze použít funkce `XChangeSaveSet` pro jeho modifikaci nebo `XRemoveFromSaveSet` pro odstranění okna.

2.17 Odchytávání chyb

Pokud dojde k příjmu nějaké chyby vzniklé používáním knihovny Xlib, je nezbytně nutné, aby správce oken takovou chybu programově obsloužil. Pokud by to neudělal, systém by přestal reagovat na další podněty. Jako parametr funkce `XSetErrorHandler` se předá ukazatel na funkci, která bude chyby zpracovávat. V této funkci se mohou definovat různá chování pro různé chybové kódy.

Kapitola 3

Analýza a návrh

V této kapitole se čtenář seznámí s rozdělením správců oken z pohledu robustnosti a ovládání. Dále bude rozebrána problematika návrhu jednotlivých částtí správce oken. Nakonec bude znázorněné rozdělení implementace do jednotlivých modulů.

3.1 Analyzování správců oken

X Window System se používá zejména na operačních systémech unixového typu. Tím, že je na unixových operačních systémech grafické uživatelské rozhraní odděleno od jádra operačního systému, je umožněna mnohem větší svoboda v navrhování grafických aplikací a správců oken. Přehled různých správců oken je možné najít na webových stránkách [5]. Rozdělení správců oken z pohledu robustnosti:

- minimalističtí správci oken (zaměřeni na rychlost a minimální výpočetní nároky počítače)
- jednoduší správci oken (nejde jim o minimaličnost, ale nenahrazují jednoduché, běžně používané aplikace)
- komplexní správci oken (snaží se nahradit i jednoduché, běžně používané aplikace)

Správce oken je možné dělit do různých skupin. Někteří správci oken se snaží být co nejméně náročni na výpočetní výkon počítače na kterém běží a nabízejí jen nejnutnější funkcionalitu. Jiní správci oken se naopak snaží svou komplexností pokrýt co největší softwarový základ a prakticky nahrazují jednoduché běžné aplikace. Rozdělení správců oken z pohledu ovládání:

- "reparentig" správci oken (orámečkovávají top-level okna aplikací)
- "tile" správci oken (dláždí plochu z top-level oken aplikací)
- správci oken u kterých je možno za běhu přepínat mezi oběmi výše uvedenými strategiemi

Z pohledu ovládání se správci oken dají rozdělit na dvě základní skupiny. Jedna skupina obaluje top-level okna aplikací rámečky. Samotná okna v tomto případě nejsou na obrazovce nijak ukotvena. Je s nimi obvykle možné po obrazovce pohybovat a mohou se také navzájem překrývat. Do rámečku okna se dále přidávají tlačítka pro manipulaci s takovým oknem. Tato tlačítka většinou plní následující funkce:

- tlačítko pro připnutí okna na všechny virtuální plochy

- tlačítko pro minimalizaci okna
- tlačítko pro maximalizaci okna
- tlačítko pro uzavření okna
- tlačítko pro změnu velikosti okna
- prostor mezi tlačítky pro posouvání okna

Druhá skupina správců oken se snaží využít maximální možnou plochu obrazovky. Okna se sebou navzájem sousedí a jejich rozměry přímo souvisí s počtem takových oken na obrazovce. Pozice oken jdou obvykle prohazovat, aby bylo možné volit podle potřeby rozměry okna. Tento způsob práce ovšem z principu neumožnuje překrývání oken a jejich plynulý posun po obrazovce.

Z pohledu uživatelů bývá právě velká rozmanitost a různá fukcionalita různých správců oken největší nevýhodou, protože často nejsou ochotni věnovat dostatek času učení se novému grafickému prostředí. Komplexní správci oken se ovšem svou podobou a ovládáním sobě navzájem velice přibližují. Problém z přechodu z jednoho takového komplexního správce oken na druhý proto nebývá tak bolestivý.

3.2 Návrh aplikace

Cílem projektu je vytvoření jednoduchého správce oken běžícího pod X Window Systémem. Implementovaný správce oken by měl mít co možná nejnižší paměťovou náročnost. Z tohoto důvodu nebude v aplikaci využito žádné vysokokoúrovňové knihovny pro realizaci prvků grafického uživatelského rozhraní. Tlačítka a nabídky budou tvořena pouze základními funkcemi knihovny Xlib.

Vytvoření rámečků

Vzhledem k tomu, že "reparenting" správci oken jsou mnohem obvyklejší než "tile" správci oken, bude implementovaný první typ. To znamená, že top-level okna aplikací budou obalována rámečky s tlačítky. Při vytvoření top-level okna aplikace se zjistí jeho rozměry. Vytvoří se okno rámečku, které bude o náležitý rozměr větší než top-level okno aplikace. Nakonec se na vhodných pozicích v rámečku vytvoří tlačítka.

Vytvoření klientů

Z každého top-level okna aplikace se musí vytvořit tzv. klient. Klient je datová struktura, která obsahuje odkazy na okno aplikace, rámeču a tlačítek, ukazatele na následujícího a přechozího klienta, flagy, zda je okno připnuté na všechny virtuální plochy nebo zda je minimalizované či maximalizované, informaci na které virtuální ploše se okno nachází, titulek okna a informace o pozici a rozměrech okna.

Ukládání klientů

Z důvodu, že je počet oken zobrazených na obrazovce v čase měnitelný a s ním i počet struktur typu `client`, je nutné takové struktury ukládat do seznamu či zásobníku. K tomuto účelu je vytvořena struktura typu `TStack`, která obsahuje ukazatele na tři klienty.

První ukazatel ukazuje na prvního klienta, což znamená, že pokud bude zobrazena plocha do které náleží, bude takové okno klienta namapované na vrcholu. Druhý ukazatel ukazuje na posledního klienta, což znamená, že pokud bude zobrazena plocha do které náleží, bude takový klient zobrazen na uplném dně. Ostatní klienti budou budou v patřičném pořadí mezi nimi. Poslední, třetí, ukazatel ukazuje na klienta, který má aktuálně přiřazen keyboard focus. Pokud na dané virtuální ploše nebude žádné okno, hodnota tohoto ukazatele bude nastavena na NULL.

Virtuální plochy

Problematika práce se strukturami typu `client` v zásobníku typu `TStack` se komplikuje v případě použití virtuálních ploch. V jediném takovém zásobníku se nachází více klientů, kteří náleží do různých virtuálních ploch, nebo jsou dokonce připlí na všechny virtuální plochy. V případě, že se uživatel přepne z jedné virtuální plochy na jinou, se prochází struktura `stack`, která je proměnnou datového typu `TStack`, směrem od posledního klienta k prvnímu. Průchod tímto směrem zajistí, že klienti mapovaní dříve jsou překrytí klienty mapovanými později a že se zanoření klientů nezmění po změně virtuální plochy a následném navrácení na původní virtuální plochu. Jednotliví klienti jsou kontrolováni na přislušnost k vybrané aktuální ploše. Pokud do virtuální plochy náleží nebo jsou jejich okna připnuta na všechny virtuální plochy, tak jsou jejich top-level okna mapována na obrazovku. V opačném případě, že jejich okna do vybrané virtuální plochy nenáleží, jsou jejich okna odmapována.

Cirkulace okny

Podobný problém nastává při cirkulování s okny na aktuální virtuální ploše. Cirkulace s okny pracuje tak, že první klient náležící do aktuální virtuální plochy se přesune v datové struktuře `stack` až na úplný konec a na jeho top-level okno se zavolá funkce, která toto okno přesune na dno obrazovky. Tím se zachová korespondence mezi strukturou `stack` stavem na obrazovce. Naopak na klienta, který se po tomto úkonu dostane pro vybranou vyrtuální plochu co nejvíce k vrcholu struktury `stack` se zavolá funkce, která zobrazí top-level okno tohoto klienta na vrchol obrazovky.

Přidělování keyboard focusu

Další věcí, kterou je třeba zajistit, je, aby byl oknům aplikací správně přidělován keyboard focus. Keyboard focus má u "reparenting" správců oken přidělené okno, které se na aktuální virtuální ploše vyskutuje na samém vrcholu. Je proto nutné naimplementovat funkci, která při každé změně zanoření oken, bude ve struktuře `stack` hledat klienta, jehož okno je aktuálně zobrazeno na vrcholu obrazovky a přidělí mu keyboard focus.

Kreslení

Kreslení a vypisování textu je řešené funkcemi z knihovny Xlib. To má za následek daleko menší paměťovou náročnost aplikace, než v případě, že by bylo využito některé vysokoúrovňové knihovny pro práci s grafickým uživatelským rozhraním. Hlavní věcí, kterou je potřeba vykreslovat, jsou okna rámečků. Vždy, když dojde k jejich úplnému nebo částečnému překrytí a následnému odkrytí nebo v případě, kdy je jim změněna velikost, je potřeba okna rámečků překreslit. Funkce, která bude rámečky překreslovat, musí nejprve zjistit rozměr rámečku a také, zda se jedná o okno s keyboard focusem, aby se vykreslovalo

správnou barvou. Následně se vybarví pozadí rámečku a barevně se zvýrazní obrysy. Nakonec je třeba vybarvit pozadí všech tlačítek v rámečku a do některých vykreslit symboly. Pokud dojte k překrytí a následnému odkrytí okna panelu, je nutné vykreslit jeho obrys znovu. Překreslování oken pluginů umístěných v panelu je zcela v jejich v režii. Při kreslení obrysu okna při posouvání okna nebo při změně jeho velikosti bude využitá logická funkce `XOR` nastavená v příslušném grafickém kontextu. Tím se zajistí možnost mazání obrysu okna aniž by bylo potřeba pamatovat si všechny hodnoty překreslovaných pixelů.

Klávesové zkratky

Pro zjednodušení práce by měly být implementovány klávesové zkratky. Klávesová zkratka bývá obvykle složena ze současného stisku jedné nebo více modifikačních kláves (Alt, Shift) a některé jiné klávesy. Informaci o takto stisknuté kombinaci kláves ovšem nesmí dostat okno aplikace, které má nastavený keyboard focus, ale správce oken. Tento stav zajistí vhodné nagrabování kláves. Správce oken následně vyvolá patřičnou činnost související s konkrétní klávesovou kombinací.

Zpracování událostí

Správce oken si vybere typy událostí a okna, od kterých chce události přijímat a to včetně oken jednotlivých klientů. Správce oken obsahuje jednu hlavní smyčku pro zpracování událostí. Pro roztřídění příchozích typů událostí je vytvořena následující struktura:

```
void (*handler[LASTEvent])(XEvent*) = {
      [KeyPress]         = keypress,
      [MappingNotify]    = mappingnotify,
      [MapRequest]       = maprequest,
      [ButtonPress]      = buttonpress,
      [Expose]           = expose,
      [MotionNotify]     = motionnotify,
      [ConfigureRequest] = configurerequest,
      [DestroyNotify]    = destroynotify,
      [UnmapNotify]      = unmapnotify,
      [PropertyNotify]   = propertynotify
};
```

Jestliže je tedy ve struktuře `handler` typ příchozí události, zavolá se jemu příslušná funkce, která má za úkol tento typ událostí zpracovávat. Každá taková funkce tedy zpracovává výhradně jeden typ událostí.

Pluginy

Pluginy jsou realizovány formou sdílených knihoven. Každý plugin dostane od správce oken přidělené jedno okno mapované do panelu. Jakou šířku bude toto okno mít a zda bude namapované na panelu zleva či zprava, se určí z konfiguračního souboru. Aby grafika pluginů ladila se zbytkem aplikace se předpokládá, že pro výběr barev budou použitá makra ze souboru `config.h`. Pro zavedení plugnů je navrhnutá struktura typu `TPluginSocket`:

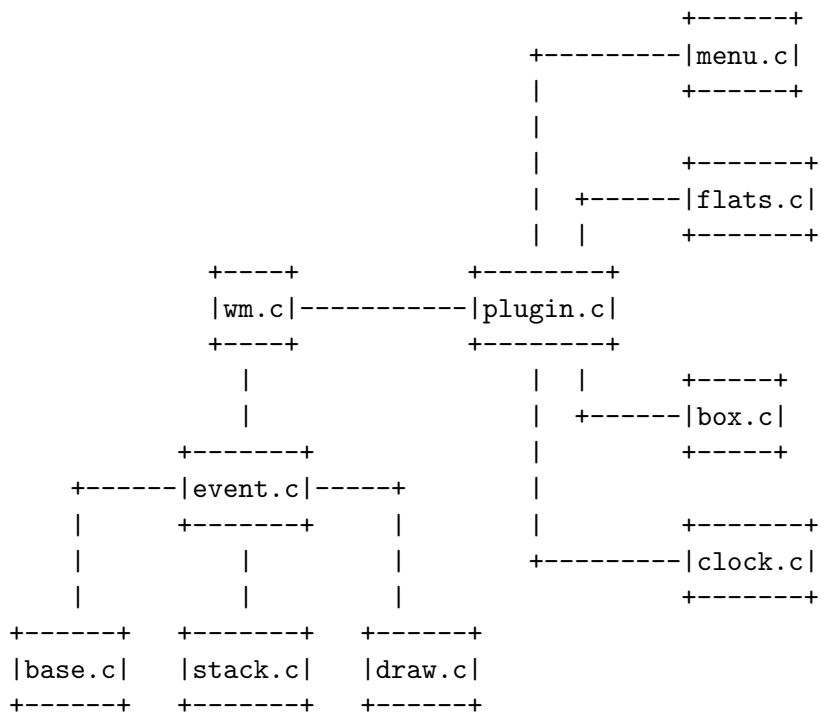
```
typedef struct {
      Window window;
      int width;
      void* handle;
      void (*function)(XEvent* xevent, int wm_event, Window w);
} TPluginSocket;
```

Prvním parametrem je okno, které plugin dostene přidělené od správce oken. Do tohoto okna se bude vykreslovat grafika pluginu, případně přes něho bude prováděna jiná interakce s uživatelem (např. klikání myší). Druhý parametr informuje plugin, jaká je šířka jemu přiděleného okna. Třetí parametr handler je potřebný k přístupu do sdílené knihovny pluginu. Posledním parametrem je potom ukazatel na funkci ve sdílené knihovně pluginu, která se stará o samotné vykonávání činnosti pluginu. Aby se pluginy mohly zavést, alokuje se nejprve pole tohoto typu `TPluginsSocket`, o rozměru závisejícím na počtu zaváděných pluginů.

Plugin potřebuje přijímat nejen události generované knihovou Xlib (např. Expose, ButtonPress, ...), ale i informace, které událostmi typu `Xevent` vyjářit nejdou. Jsou to například informace o cirkulaci oken klientů, zrušení klienta, přepnutí virtuální plochy, časovací informace a další. K tomuto účelu jsou ve správci oken nadefinovány vlastní události typu `int`. Hlavní funkce pluginu má dva parametry. Jeden je typu `XEvent`, což je událost od knihovny Xlib a druhý je typu `int`, což je událost zasílaná správcem oken. Plugin podle hodnot těchto parametrů pozná, kterou činnost má vykonat.

Rozdělení implementace do modulů

Implementované funkce správce oken jsou rozdělené do modulů podle činnosti, kterou vykonávají. Popis jednotlivých pluginů a jejich funkcí popisuje následující kapitola 4. Následující schéma zjednodušeně znázorňuje závislosti jednotlivých modulů a pluginů.



Kapitola 4

Implementace a testy

V této kapitole bude popsána implementace jednotlivých částí správce oken. Implementace je rozdělena do specializovaných modulů. Z každého modulu zde budou uvedeny všechny jeho implementvané funkce a případně i datové struktury. Implementované funkce budou popisovány obecně zjednodušenou formou, důležité části budou popsány podrobněji. Při implementaci je vhodné čerpat informace z knihy Xlib Reference Manual for Version 11 [2], kde jsou přehledně popsány jednotlivé funkce knihovny Xlib. Podobný přehled funkcí je možné najít i na webových stránkách [4]. Implementace správce oken je rozdělena do následujících modulů:

- wm.c
- base.c
- event.c
- stack.c
- draw.c
- plugin.c

Funkcionalitu správce oken lze dále rozšiřovat pomocí pluginů, které jsou ve formě sdílených knihoven. Prozatím implementované pluginy jsou:

- menu.c
- flats.c
- box.c
- clock.c

4.1 Modul wm.c

V modulu wm.c se nachází funkce `main` a hlavní smyčka událostí. Ve funkci `main` dochází zejména k nastavení základních datových struktur správce oken. Nejprve se modul pokusí připojit voláním funkce `XOpenDisplay` k X serveru. Jestliže se to nepodaří, správce oken se bezprostředně ukončí s vypsáním chybové hlášky na standartní chybový výstup.

Funkcemi `XDisplayWidth` a `XDisplayHeight` se určí rozlišení obrazovky. Voláním funkce `XCreateCursor` se vytvoří tvary kurzorů a uloží se do pole. Jeden tvar kurzoru je pro normální práci (šipka), druhý pro přesouvání top-level okna (kříž) a třetí pro změnu velikosti top-level okna (úhelník).

Jako výchozí se nastaví první virtuální plocha a inicializuje se struktura `stack` což je proměnná typu `TStack`. Root oknu se nastaví příjem požadovaných událostí a nastaví se mu zobrazování požadovaného tvaru kurzoru - šipky. Dále se vytvoří okno představující panel. Toto okno má nastavený bit `override_redirect`, aby nedošlo ke spravování tohoto okna správcem oken. Dále je vytvořeno a nastaveno několik grafických kontextů, které jsou využívány i v jiných modulech. Volají se funkce pro nagrabování klávesových zkratek, načtení pluginů a nalezení již existujících top-level oken.

Významnou částí tohoto modulu je hlavní smyčka událostí. Události jsou podle svých typů rozesílány funkcím, které tyto událostí zpracovávají. Každá událost je též rozeslána všem pluginům, přičemž záleží na pluginu, zda událost zpracuje.

4.2 Modul base.c

V modulu base.c se nalézají základní funkce pro činnost správce oken, které svým zaměřením souvisí s různými problematikami. Není vhodné je proto zařadit do jiných modulů.

Vypisování chyb

Pokud je nutné z nějakého důvodu bezprostředně ukončit správce oken, je volána funkce `error_exit`. Funkci je předány formátovaný textový řetězec, který specifikuje vzniklou chybu. Funkce vypíše tento formátovaný textový řetězec na standardní výstup a ukončí běh správce oken.

Spouštění programů

Samozřejmou činností správce oken je spouštění programů. Přinejmenším musí umožnit spuštění aplikace `xterm`, s jejíž pomocí se dají spouštět další aplikace. Funkce, která spouštění aplikací provádí, má název `start_program`. Jejím parametrem je řetězec značící jméno programu, který chceme spustit. Vytvoření nového procesu je realizováno funkcí `vfork` a funkce `execl` zajistí provádění požadovaného programu.

Grabování klávesových zkratek

Ve funkci `grab_hotkeys` jsou definované kombinace kláves, které mají ve správci speciální funkčnost. Jsou to kombinace kláves pro přepínání virtuálních ploch, přesouvání top-level oken na jiné virtuální plochy, cirkulování top-level oken, spuštění aplikace `xterm` a kombinace kláves pro ukončení správce oken. Každá kombinace kláves je nagrabována funkcí `XGrabKey` s příslušnými modifikátory (Alt a případne Alt + Shift).

Vytvoření klienta

Pokud je vytvořeno top-level okno aplikace, je volána funkce `make_client`, která pro toto top-level okno naplní strukturu `client`. Dříve než je top-level oknu změněn předek z root okna na okno rámečku, je původní top-level okno přidáno do seznamu zvaného `save-set`.

V paměti se alokuje prostor pro strukturu `client`. Zjistí se rozměry původního top-level okna, které jsou i s flagy `min` a `max` uloženy do struktury `client`. Původní orámování top-level okna je nežádoucí a proto je zrušeno nastavením šířky orámování na 0 pixelů. Dále se vytvoří všechna tlačítka rámečku okna a funkci `XSelectInput` je jim vybraný příjem událostí typu `ButtonPress`. U okna rámečku je vybraný i příjem události typu `Exposure`, aby jej šlo případně překreslit. Všechna takto vytvořená tlačítka i s oknem aplikace změní svého předchůdce z root okna na okno rámečku voláním funkce `XReparentWindow`. Následně se všechna okna namapují. Funkcí `XFetchName` se zjistí titulek aplikace, který se rovněž uloží do struktury client. Poslední činností funkce `make_client` je vložení takto vytvořené struktury `client` do struktury `stack`.

Posun top-level okna po obrazovce

Posun top-level oken po obrazovce zajišťuje funkce `move`. Funkcí `XQueryPointer` se zjistí souřadnice, na kterých došlo ke stisku tlačítka myši. Zjistí se rozměry okna aplikace. Tvar kurzoru se změní na tvar značící přesouvání. Funkcí `XChangeActivePoiterGrab` se změní parametry grabování myši a funkcí `XGrabServer` se zamezí příjem událostí od ostatních klientů. Následuje nekonečná smyčka událostí zpracovávající události typů `MotionNotify` a `ButtonRelease`.

`Přijde-li` událost typu `MotionNotify`, je pomoci flagu `box_drawn` zjištěno, `zdali` je vykreslený obrys okna. Pokud je tomu tak, zavolá se funkce `undraw_box`, která obrys okna zruší. Funkcí `XQueryPointer` se opět zjistí aktuální pozice kurzoru na obrazovce. Z této pozice je možné vypočítat umístění okna a funkcí `draw_box` vykreslit nově umístěný obrys.

`Přijde-li` událost typu `ButtonRelease`, je pomocí flagu `box_drawn` zjištěno, zda je vykreslený obrys okna. Pokud je tomu tak, zavolá se funkce `undraw_box`, která obrys zruší. Funkcí `XQueryPointer` se naposledy zjistí konečné udaje o pozici a funkcí `XMoveWindow` se provede přesun okna.

Změna velikosti top-level okna

Jetliže je potřeba změnit velikost top-level okna, je volána funkce `resize`. Nejprve se voláním funkce `XQueryPointer` zjistí souřadnice, na kterých došlo ke stisku tlačítka. Voláním funkce `XGetNormalHints` se zjistí flagy, které určují, `zda-li` se má oknu vůbec umožnit změna velikosti a zda má okno nastavenou nějakou minimální velikost. Funkcí `XChangeActivePointerGrab` se změní parametry grabování myši a funkcí `XGrabServer` se zamezí příjem událostí od ostatních klientů. Podobně jako ve funkci `move` je i zde použito nekonečné smyčky pro zpracování událostí typů `MotionNotify` a `ButtonRelease`.

Po příchodu události typu `MotionNotify` se pomocí flagu `box_drawn` zjistí, zda je obrys okna vykreslený. Pokud je obrys okna vykreslený, je voláním funkce `undraw_box` smazán. Následně je voláním funkce `XQueryPointer` zjištěna aktuální pozice kurzoru myši. Z této pozice se spočítají aktuální rozměry obrysu okna. Rozměry obrysu okna jsou v závislosti na flagách `width_inc` a `height_inc` přepočítány na nejbližší dovolenou velikost. Následně je volána funkce `draw_box`, která vykreslí obrys okna o požadované velikosti.

Přijetí události `ButtonRelease` značí dokončení procesu změny velikosti top-level okna. `Je-li` nastaven flag `box_drawn`, odstraní se voláním funkce `undraw_box` obrys okna. Voláním funkcí `XResizeWindow` je změněna velikost rámečku okna a velikost okna aplikace. Následuje volání funkcí `XMoveWindow`, kterými se přesunou okna tlačítek v rámečku na vhodné pozice.

Maximalizace top-level okna

Funkce `miximize` zjistí, zda je nastavený flag `max`. Pokud je flag `max` nastavený, znamená to, že je okno roztažené přes celou obrazovku. Funkce `maximize` v tomto případě zmenší okno voláním funkce `XResizeWindow` zpět do původních rozměrů. Původní rozměry zjistí z proměnných `width` a `height` ze struktury `client`. Dále je nutné voláním funkcí `XMoveWindow` přesunout okna tlačítek umístěných v rámečku na správné pozice. Nakonec je flag `max` vynulován.

V druhém případě, kdy není flag `max` nastavený, je nutné okno roztáhnout na rozměry obrazovky. Rozměry obrazovky jsou známé z globálních proměnných `display_width` a `display_height`. Následně se posunou okna tlačítek rámečku na správné pozice a nastaví se flag `max`.

Minimalizace top-level okna

Funkce `minimize` zjistí, zda je nastavený flag `min`. Pokud je flag `min` nastavený, znamená to, že je okno rámečku zmenšené na velikost horní lišty. Funkce `minimize` tedy zvětší okno rámečku do původní velikosti a vynuluje flag `min`. Původní rozměry zjistí z proměnných `width` a `height` ve struktuře `client`. Pokud však flag `min` nastavený není, znamená to, že má okno rámečku původní rozměry. Proto funkce `minimize` zjistí šířku okna aplikace a okno rámečku zmenší, aby byla zobrazena jen horní lišta. Posléze je nastaven flag `min`.

Dealokace zdrojů

Při ukončení běhu správce oken je volána funkce `cleaning`, která uvolní alokované zdroje. Nejdříve je volána funkce `unload_plugins`, která odstraní z panelu pluginy a uvolní paměť jimi alokovanou. Dále se uvolní všechny použité grafické kontexty a buffer, do kterého správce oken ukládá řetězce ze standardního vstupu. Keyboard focus se nastaví na root okno.

Následně se v cyklu načítají jednotliví klienti ze struktury `stack`. Z každé struktury `client` je nejprve funkcí `XKillClient` zrušeno top-level okno aplikace. Potom jsou funkcí `XDestroySubwindows` zrušena všechna okna, která jsou následníky okna rámečku. Teprve potom je zrušeno funkcí `XDestroyWindow` okno rámečku. Zbývá dealokovat paměť použitou pro titulek, odstranit strukturu `client` ze struktury `stack` a dealokovat paměť použitou pro strukturu `client`.

Nalezení top-level oken při startu

Po spuštění správce oken se musí zavolat funkce, která zjistí, zda na root oknu nejsou namapovaná nějaká top-level okna. V případě, že taková okna existují, se z nich vytvoří klienti. Funkce k tomuto učelu určená se nazývá `discover_topwindows`. Tato funkce volá funkci `XQueryTree`, která vráti ukazatel na pole všech top-level oken a jejich počet. V cyklu se pro každé takové top-level okno volá funkce `search_topwindow`, která podle přítomnosti tohoto okna ve struktuře `stack` vrátí odpovídající pravdivostní hodnotu, zda se tam již dané okno nevyskytuje. Pokud tedy top-level okno neodpovídá žádnému klientu ve struktuře `stack` a nejedná se o top-level okno s flagem `override_redirect`, je mu vytvořený nový klient, který je v zápětí vložený do struktury `stack`.

Odchytávání Xlib chyb

Pro zpracování chyb generovaných knihovnou Xlib je třeba naprogramovat tzv. error handler. Ve správci oken tuto činnost řeší funkce `xlib_error`. Pro jednoduchost je postačující, když tato funkce pouze vrací celočíselnou hodnotu.

4.3 Modul event.c

Modul `event.c` obsahuje funkce, které výhradně zpracovávají události podle jejich typů. Každá funkce se specializuje na jeden jediný typ události. Události jsou těmto funkcím zasílány z hlavní smyčky událostí umístěné v modulu `wm.c`. Názvy funkcí v tomto modulu jsou shodné s názvy typů událostí.

keypress

Událost typu `KeyPress` ve své struktuře obsahuje `keycode`. `Keycode` se převede na `keysym`, který je spolu s maskami (Shiftu, Altu, NumLocku) porovnáván na shodu s některou klávesovou kombinací. Pokud dojde ke shodě, provede se odpovídající činnost, která souvisí s danou klávesovou zkratkou. To znamená zejména přepínání virtuálních ploch, přeposílání klientů na jiné virtuální plochy, cirkulace klientů na aktuální virtuální ploše, spuštění xtermu a ukončení běhu správce oken.

maprequest

Příchod události typu `MapRequest` znamená, že se nějaké top-level okno chce namapovat na obrazovku. Nejprve je tedy nutné zjistit, zda má nastavený flag `override_redirect`. Pokud je tento flag nastavený, provádění funkce se bezprostředně ukončí, protože toto top-level okno si nepřeje být spravované správcem oken.

V opačném případě se volá funkce `make_client`, které je předán odkaz na top-level okno, ze kterého se následně vytvoří klient. Následuje funkce `search_focus_window`, která tomuto klientu zpravidla přidělí keyboard focus a funkce `distribute_plugins`, která bude informovat pluginy o vytvoření nového klienta.

buttonpress

Činností funkce `buttonpress` je zjistit, zda došlo ke stisku některého tlačítka v rámečku některého z top-level oken a v případě, že k této s kutečnosti došlo, provést patřičnou akci. U každé události typu `BottonPress` je tedy v cyklu pro každého klienta zkoumané, jestli okno, ve kterém došlo ke vzniku události, je totožné s některým oknem v jeho struktuře. Pokud ke shodě došlo, provede se pro tohoto klienta jedna z rutin: připnutí top-level okna na všechny plochy, minimalizece, maximalizece, uzavření nabo změna velikosti top-level okna.

expose

Funkce `expose` zajišťuje překreslení oken, které byly uplně nebo častečně překryty jiným oknem. Nejprve se testuje, zda bylo překryto okno znázorňující panel. Jestliže bylo překryto, je okno panelu překresleno funkcí `redraw_panel`. Dále se v cyklu pro všechny klienty testuje, zda je nutné překreslit orámování okna. `Je-li` to nutné, tak se dále zjistí, jestli se

jedná o klienta, který má nastavený keyboard focus, aby se rámeček překreslil vhodnou barvou. Nakonec se zavolá funkce `redraw_client` pro překreslení okna.

configurerequest

Událost typu `ConfigureRequest` v sobě nese informace, jak se chce okno aplikace překonfigurovat. Parametry `x`, `y`, `width`, `height`, `sibling` a `stack_mode` jsou překopírované do struktury typu `XWindowChanges`, která je posléze předána funkci `XConfigureWindow`, která změnu konfigurace okna provede. V cyklu se zjistí, které okno aplikace bylo překonfigurované a upraví se velikost rámečku.

destroynotify

Událost typu `DestroyNotify` informuje o zrušení okna aplikace. Funkce `destroynotify` proto prochází v cyklu všechny klienty a zjišťuje, zda rušené okno není totožné s oknem aplikace některého z klientů. Pokud se zjistí, že zrušené okno skutečně odpovidá některému oknu aplikace klienta, jsou funkcí `XDestroySubwindows` zrušena všechna okna v rámečku takového klienta. Dále se funkcí `XDestroyWindow` zruší i samotné okno rámečku. Posléze se dealokují paměťové zdroje zabrané rušeným klientem. Funkcí `search_focus_window` se přiřadí keyboard focus vhodnému oknu a funkcí `distribute_plugins` se informují pluginy, že došlo k odstranění klienta.

unmapnotify

Po příchodu události typu `UnmapNotify` se v cyklu zjistí, zda okno, které se odmapovalo, je oknem aplikace některého z klientů. Jestliže tomu tak je, tak se voláním funkce `XDestroyWindow` zruší okna tlačítek rámečku. Předchůdce okna aplikace se změní z okna rámečku na root okno a okno rámečku se rovněž zruší voláním funkce `XDestroyWindow`. Následně se dealokují paměťové zdroje zabrané rušeným klientem. Funkcí `search_focus_window` se přiřadí keyboard focus vhodnému oknu a funkcí `distribute_plugins` se informují pluginy, že došlo k odstranění klienta.

propertynotify

Pokud došlo ke změně property je generováná událost typu `PropertyNotify`. V cyklu se zjistí, zda property měnilo některé top-level okno aplikace a zda se jedná o atom `XA_WM_NORMAL_HINTS`. V případě, že ano, je funkcí `XGetWMNormalHints` naplněna struktura typu `XSizeHints` a posléze změněny rozměry rámečku okna.

4.4 Modul stack.c

Každé top-level okno aplikace je spolu s dalšími informacemi uloženo správcem oken při jeho vytvoření do struktury `client`. Nutnost spravovat různý počet takových struktur vede k vytvoření dynamické datové struktury, která bude umožňovat procházet a modifikovat tyto struktury `client`.

Struktura pro uložení klientů

Modul `stack.c` k tomuto účelu definuje strukturu `TStack` obsahující 3 členy: ukazatel na prvního a posledního klienta v seznamu a ukazatel na klienta, který má keyboard focus.

```
typedef struct {
    client* top;            // nejhornější klient
    client* bottom;         // nejspodnější klient
    client* focus;          // klient mající keyboard focus
} TStack;
```

Struktura `stack` je navržena tak, že míra zanoření klientů ve struktuře odpovídá míře zanoření top-level oken na obrazovce. Je nutno podotknout, že tento proces bere v potaz jen ty klienty, kteří mají nastavenou příslušnost k aktuální virtuální ploše nebo klienty, kteří jsou připnutí na všechny virtuální plochy. Klienti, kteří mají nastavenou příslušnost k jiné virtuální ploše, než je právě nastavená, jsou ignorováni.

Vložení klienta

Předtím, než je nově vytvořené top-level okno aplikace namapováno, je naplněna struktura `client`. Funkce `insert_client` umístí takto naplněnou strukturu na vrchol datové struktury `stack`. Umístění struktury `client` na vrchol struktury `stack` zajistí, že takto nově vytvořené top-level okno bude na obrazovce namapováno na vrcholu.

Odstranění klienta

Jestliže je některé top-level okno rušeno, je nutné odstranit i patřičný záznam ze struktury `stack`. K tomu slouží funkce `delete_client`. Funkce `delete_client` nejprve zjistí, zda mazaný klient nemá nastavený keyboard focus. Pokud ho nastavený má, tak člen `stack.focus` nastaví na `NULL`. Dále funkce zjišťuje, na které pozici se klient ve struktuře `stack` nachází (osamoceně, na vrcholu, na dně nebo uvnitř) a podle toho klienta ze struktury `stack` vhodným způsobem odstraní.

Cirkulace top-level okny

Obvyklou činností správců oken je cirkulování s top-level okny a to pro každou virtuální plochu zvlášť. Top-level okno, které je namapováno na obrazovce na vrcholu se přesune na úplne dno. Okno, které bylo pod ním, se tak dostane na vrchol.

Složitější situce ovšem panuje v datové struktuře `stack`, kde se kromě klientů patřících na danou virtuální plochu vyskytují i klienti patřící do jiných virtuálních ploch. Abychom mohli s top-level okny cirkulovat, je nutné nalézt ve struktuře `stack` alespoň dva klienty náležící k dané ploše. První klient pro danou vituální plochu nejblíže k vrcholu struktury `stack` koresponduje s nejhornějším top-level oknem na obrazovce. Někde dále ve struktuře `stack` je nutné najít druhého klienta s danou virtuální plochou, jehož top-level okno bude posléze na obrazovce namapováno na vrchol. Funkce `circulate_topwindows` nejprve nalezne oba dva klienty, aby mohla začít se samotnou cirkulací. První klient se z jeho dosavadní pozice ve struktuře `stack` odstraní voláním funkce `delete_client` a voláním funkce `insert_client_bottom` se vloží na konec struktury `stack`. Funkce `XRaiseWindow` nakonec

zobrazí top-level okno horního klienta na vrchol obrazovky a funkce `XLowerWindow` přesune okno původně horního klienta na samé dno obrazovky.

Přepínání virtuálních ploch

Funkce `set_flat` prochází celou strukturou `stack`. Pro každého klienta, který je v ní obsažený, zkontroluje, zda se má jeho top-level okno na dané virtuální ploše zobrazit. Pokud ano, tak ho funkcí `XMapWindow` namapuje a funkce `XRaiseWindow` zajistí jeho vhodné zanoření. Pokud ne, tak je jeho top-level okno odmapováno funkcí `XUnmapWindow` a to i tehdy, pokud okno není namapované. Volání `XUnmapWindow` na nenamapované okno totiž nemá žádný vliv. Směr průchodu ode dna směrem k vrcholu strukturou `stack` zajistí, že se top-level okna namapují na obrazovku v patřičném pořadí.

Posílání top-level oken na jiné virtuální plochy

V některých případech je vhodné přeposlat top-level okno z jedné virtuální plochy na jinou. K této činnosti slouží funkce `send_client`. Pokud klient nemá nastavený flag `clip` a `nepřeposíláme-li` klienta na aktuální virtuální plochu, funkce odmapuje patřičné top-level okno a změní obsah proměnné určující sounáležitost k virtuální ploše na hodnotu určující číslo virtuální plochy, na kterou chceme top-level okno přeposlat. Funkcemi `raise_client` a `XRaiseWindow` se zajistí, že se top-level takového klienta zobrazí na vrcholu virtuální plochy na kterou byl přeposlán.

Připínání top-level oken na všechny virtuální plochy

Někdy je vhodné mít možnost připnout top-level okno na všechny virtuální plochy. K tomuto účelu slouží funkce `clip_client`. Funkce nejprve zjistí, jestli má klient nastaven flag `clip`. Tento flag nese informaci, jestli je top-level okno připnuté na všechny virtuální plochy. Pokud tento flag nastaven není, funkce ho nastaví. V opačném případě funkce `clip_client` flag `clip` vynuluje a nastaví číslo virtuální plochy na hodnotu aktuální virtuální plochy. To způsobí, že okno, které bylo původně připnuté na všechny virtuální plohy, zůstane po odepnutí pouze na virtuální ploše, ve které došlo k odepnutí. Nakonec funkce `clip_client` volá funkci `raise_client`, která přemístí klienta na začátek struktury `stack`, aby se i po přepnutí na ostatní plochy namapovalo jeho top-level okno na vrcholu.

Hledání top-level okna s keyboard focusem

Potom, co jsou užity funkce, které ovlivňují zobrazení top-level oken nebo jejich zanoření na obrazovce, je nutné volat funkci `search_focus_window`, která ve struktuře `stack` určí aktuální top-level okno a přiřadí mu keyboard focus. Aktuální okno se určí prohledáváním structury `stack` od vrcholu směrem ke dnu. Vybere se první okno, které má nastavený flag `clip` nebo které náleží k aktuálně vybrané aktuální virtuální ploše. Funkce `search_focus_window` takovému oknu přiřadí keyboard focus a nastaví na něho ukazatel `stack.focus`.

Zjišťování, zda je z top-level okna klient

Při spuštění správce oken je dobré prozkoumat, jestli již neexistují top-level okna. Pokud taková okna existují, je možné, že z nich nejsou vytvořeni klienti a nejsou tudíž uloženi

ani ve struktuře `stack`. Ke zjištění, zda je takové top-level okno klientským oknem ve struktuře `stack`, slouží funkce `search_topwindow`. Tato funkce projde celou strukturu `stack` a zjišťuje, jestli se v ní dané top-level okno nachází. Podle toho, zda se okno ve struktuře našlo nebo nenašlo, vrátí odpovídající pravdivostní hodnotu.

Překreslování rámečků

Určit, které okno má oktuálně přiděleny keyboard focus, lze pomocí barvy rámečku. Jediné top-level okno s keyboard focusem má odlišnou barvu, než okna, která keyboard focus nemají. Specifikovat tyto barvy lze modifikací souboru `config.h`. Činnostmi jako je cirkulace top-level oken, vytvoření a zrušení nového top-level okna, přepnutí virtuální plochy nebo kliknutím do některého top-level okna, dochází zpravidla ke změně přidělení keyboard focusu. Rámeček top-level okna, které získalo keyboard focus se zvýrazní speciální barvou. Jestliže existovalo top-level okno, které keyboard focus ztratilo, je nutné vybarvit jeho rámeček běžnou barvou.

K zajištění správného vybarvování rámečků top-level oken při změnách keyboard focusu slouží funkce `redraw_clients`. Funkce `redraw_clients` prochází všechny záznamy struktury `stack`. Jestliže na klienta této struktury ukazuje ukazatel `stack.focus`, je na tohoto klienta volána funkce `redraw_client` s parametrem značícím, že se má rámeček vybarvit zvýrazňující barvou. V opačném případě, kdy na daného klienta ukazatel `stack.focus` neukazuje, se na něho opět zavolá funkce `redraw_client`, ovšem s parametrem značícím, že se má rámeček vybarvit běžnou barvou.

4.5 Modul draw.c

Modul `draw.c` obsahuje funkce související s kreslením, vykreslováním a alokováním příslušných struktur.

Získání barvy

Mnoho kreslících funkcí knihovny Xlib vyžadují jako parametr předat hodnotu pixelu značící barvu. Funkci `pixel_value` je předán řetězec vyjadřující barvu v klasickém 24bitovém RGB tvaru (např. #00FF00 pro zelenou barvu). Funkcí `XAllocNamedColor` je posléze alokována barva. Funkce `pixel_value` z alokované struktury barvy vrátí hodnotu pixelu této barvy.

Překreslování rámečků top-level okna

`Dojde-li` k překrytí a následovnému odkrytí rámečku nebo jeho části, je nutné rámeček znovu překreslit. K překreslení rámečku slouží funkce `redraw_client`. Funkce nejprve zjistí rozměry okna aplikace. Podle jí předaného parametru focus, který značí, zda jde o top-level okno s keyboard focusem, nastaví barvu rámečku. Následuje vybarvení plným obdélníkem okna rámečku a všech tlačítek umístěných v horní liště. V dalším kroku se změní barva na barvu, kterou se vykreslují vzory na tlačítkách. V dolních rozích se plným obdélníkem vykreslí okna pro změnu velikosti top-level okna. Zjistí se, jestli se jedná o top-level okno, které je připnuto na všechny virtuální plochy a vykreslí se do tlačítek patřičné grafické vzory. V posledním kroku se změní barva na černou. Touto barvou se obtáhne celé okno rámečku, vykreslí se linky mezi oknem aplikace a horní a dolní lištou a v poslední řadě se obtáhnou tlačítka pro změnu velikosti top-level okna.

Kreslení ohraničení okna

Při změně velikosti top-level okna nebo při jeho posouvání po obrazovce správce oken vykresluje jeho obrys. Tím se přesně určí pozice a velikost top-level okna v každém okamžiku, kdy je s ním manipulováno. Obrys okna lze vykreslit voláním funkce `draw_box`. Funkce vezme jako parametry souřadnice horního levého a dolního pravého rohu obrysu. `Je-li` třeba obrys top-level okna odstranit, zavolá se funkce `undraw_box`. Tato funkce se liší od funkce `draw_box` pouze jménem. Parametry a tělo mají obě funkce z principu logické funkce `XOR` stejné. Aby došlo ke korektnímu odstranění obrysu top-level okna, musí být funkce `undraw_box` volána se stejnými parametry jako byla předtím volána funkce `draw_box`. Grafický kontext, kterým se obrys top-level okna vykresluje, musí mít nastavenou funkci `GXxor`, aby nebylo nutné si pamatovat hodnoty všech pixelů, které jsou překreslovány.

Překreslení panelu

`Dojde-li` při přesunování některého top-level okna k překrytí a následovnému odkrytí části panelu, je nutné panel překreslit. O překreslování panelu se stará funkce `redraw_panel`. Po jejím zavolání vyplní panel barvou definovanou v souboru `config.h`. Posléze obtáhne horní a boční části panelu černou linkou. Překreslování oken pluginů umístěných v panelu probíhá autonomě v jejich režii.

4.6 Modul plugin.c

Události, které jsou pluginům rozesílány, jsou dvou různých typů. Prvním typem jsou klasické události knihovny Xlib, tzn. typ `XEvent`. Tento typ událostí je zasílán přímo z hlavní smyčky událostí. Druhým typem jsou události typu `int`, které jsou nadefinované správcem oken a oznamují události, které nelze vyjádřit klasickými událostmi `XEvent` (změna virtuální plochy, cirkulace oken na ploše, odstranění klienta, ...). Kvůli potřebě uchovávání dat souvisejících s pluginy byla navržena struktura typu `TPluginSocket`. Struktura obsahuje okno, které je danému pluginu přiděleno, jeho šířku, handle do sdílené knihovny a ukazatel na funkci pluginu.

Zavádění pluginů

K zavedení pluginů do správce oken se používá funkce `load_plugins`. Configurace pluginů se nachází v souboru `plugins/plugins.conf`. Aby se dalo z tohoto souboru číst, je nutné zjistit absolutní cestu k tomuto souboru. Absolutní cesta se složí z nultého parametru příkazového řádku a řetězce “plugins/plugins.conf”. Soubor plugins.conf se otevře pro čtení a zjistí se počet pluginů, které se mají načíst. Alokuje se pole typu `TPluginSocket` a velikosti počtu načítaných pluginů. V cyklu se pro každý plugin načítají informace o názvu pluginu, šířce okna na panelu, které potřebuje pro svůj běh a informace, zda okno pluginu bude umístěno na panelu vlevo či vpravo. Pokud už pro aktuálně načítaný plugin není na panelu místo, správce oken se ukončí a vypíše na standardní chybový výstup příčinu ukončení.

Na patřičné pozici se vytvoří okno pluginu, přičemž odkaz na toto okno je uloženo do pole typu `TPluginsSocket`. Oknu se vybere voláním funkce `XSelectInput` příjem událostí typů `ButtonPressMask` a `ExposureMask`. Okno se namapuje voláním funkce `XMapWindow`. Volání funkce `dlopen` umožní přístup do sdílené knihovny pluginu.

Rozesílání událostí pluginům

K rozesílání událostí pluginům slouží funkce `distribute_plugins`. Tato funkce má dva parametry, prvním je ukazatel na událost `XEvent` a druhý je událost generovaná správcem oken typu `int`. Funkce `distribute_plugins` volá v cyklu všechny funkce pluginů a předává jim tyto události a okno, do kterého mají pluginy vykreslovat.

Odinstalování pluginů

Odinstalování pluginů se děje voláním funkce `unload_plugins`. V cyklu se pro každý plugin zavolá funkce `XDestroyWindow`, která zruší okno přiřazené konkrétnímu pluginu. Dále je zavolána funkce `dlclose` pro odstranění reference ke sdílené knihovně pluginu. Nakonec se uvolní paměť zabraná strukturou `plugin_socket`.

4.7 Pluginy

Pro rozšíření základních funkcí správce oken je použito pluginů. Pluginy jsou implementovány jako sdílené knihovny. Soubor `wm/plugins/plugins.conf` slouží ke konfiguraci pluginů. Pro každý plugin je vyhrazen jeden řádek v tomto souboru, který se skádá ze tří částí.

```
název_souboru_pluginu  šířka_pluginu_na_panelu  umístění_vlevo/vpravo
  - textový řetězec      - celé číslo             - 1 nebo 0
```

Plugin menu

Plugin menu slouží k rychlému spouštění programů. Menu se aktivuje kliknutím na tlačítko v panelu. Nejprve se zobrazí okno kategorií programů. Po přesunutí nad požadovanou kategorii se zobrazí okno s titulky programů. Po kliknutí na vybraný titulek programu se okna nabídek uzavřou a spustí se požadovaná aplikace.

Uživatel může v souvislosti s tímto pluginem kliknout buďto na tlačítko v panelu, nabídku kategorií nebo nabídku programů. Pokud tedy přijde událost typu `ButtonPress`, zjistí se, do kterého z těchto oken bylo kliknuto a případně v jakém stavu se nabídky nacházely. Pokud je kliknuto na tlačítko v panelu, vytvoří se okno nabídky kategorií s nastaveným flagem `override_redirect`. Pokud dojde ke stisku tlačítka na panelu a okno kategorií je již zobrazeno, dojde k jeho zrušení funkcí `XDestroyWindow`. Při stisku tlačítka myši v okně nabídek programů se z pozice, na kterou se kliklo, určí textový řetězec určující název programu a funkcí `start_program` dojde ke spuštění programu.

Přijde-li událost `ButtonPress`, která nevznikla v žádném oknu souvisejícím s pluginem, dojde ke zrušení všech nabídek funkcí `XDestroyWindow`. Další událostí, na niž plugin reaguje, je `MotionNotify`. Při posouvání kurzoru po nabídkách se zjišťuje, zda se přejelo z jednoho titulku na druhý a případně se nastavují flagy signalizující tuto skutečnost. V případě, že se kurzor přesune v nabídce kategorií z jednoho titulku na jiný, je nutné odstranit stávající nabídku programů a vytvořit novou. Při přijetí události typu `Expose` se jen nastaví flag, který signalizuje, že je nutné grafiku pluginu překreslit.

V případě, kdy je nutné grafiku pluginu překreslit, ať už z důvodu přijetí události typu `Expose` nebo z důvodu pohybu kurzoru, dojde k testování, které nabídky jsou zobrazeny a na které pozici se nachází kurzor. Podle těchto kritérií dochází k překreslování.

Samotné názvy kategorií a programů jsou uložené v poli. V jiném poli jsou uloženy indexy, na kterých se vyskytují názvy kategorií. Modifikováním těchto dvou polí můžeme modifikovat zobrazované nabídky.

Plugin flats

Plugin flats graficky znázorňuje aktuální stav na všech virtuálních plochách. Ohraničením okénka dané virtuální plochy je znázorněno, která virtuální plocha je aktuálně zobrazena. Pokud je na některé virtuální ploše umístěno jedno nebo více top-level oken, je tento stav vyznačen barevným vyplněním příslušného okénka virtuální plochy.

Plugin flats reaguje na události tupů `Expose` a `ButtonPress`. Dále musí reagovat na události generované správcem oken, které signalizují, jestli byl některý klient vytvořen, smazán, přeposlán na jinou virtuální plochu nebo připnut na všechny virtuální plochy. Plugin také reaguje na událost informující o změně virtuální plochy.

Nejdříve je nutné nastavit potřebné proměnné grafického kontextu, grafický kontext vytvořit a vyplnit pozadí okna určeného pro plugin. Pokud uživatel klikl do okna pluginu, přijde událost typu `ButtonPress`. Z této události se zjistí, na jakou pozici uživatel přesně klikl a potažmo se vypočítá, na kterou virtuální plochu si uživatel přál přepnout. Na tuto virtuální plochu se následně přepne. Dále je vytvořeno boolovské pole o rozměru počtu virtuálních ploch, které nese informace o tom, zda se na dané virtuální ploše vyskytuje jedno nebo více top-level oken. Toto pole se celé nejprve inicializuje na hodnoty False signalizující, že se na žádné virtuální ploše žádné top-level okno nevyskytuje.

V cyklu se postupně procházejí všechny záznamy struktury `stack`. Pro každý existující klient náležící k určité virtuální ploše se nastaví příslušný bit pole na hodnotu True. Pokud se však v této struktuře nalezne klient, který je připnutý na všechny virtuální plochy, tak se tato skutečnost nastaví do proměnné `all` a procházení této struktury se ukončí, protože je to zbytečné. Jestliže je proměnná `all` nastavena na hodnotu True, tak se všechny bity v poli nastaví také na hodnotu True. Dále se v cyklu prochází toto pole a pro každou proměnnou nastavenou na True se vybarví příslušné okénko virtuální plochy. Zároveň se na příslušná místa vypisují čísla virtuálních ploch.

plugin box

Plugin `box` zobrazuje v liště titulky oken pro aktuálně vybranou virtuální plochu. Titulky jsou seřazeny podle míry zanoření oken.

Plugin `box` reaguje na události typů `Expose` a `ButtonPress`. Dále na události generované správcem oken, které potřebuje pro svou činnost. Nejprve se nastaví patřičné proměnné grafického kontextu, aby se okno vyhrazené pro plugin mohlo vybarvit. V cyklu se projdou všechny záznamy struktury `stack`, z čehož se zjistí, kolik klientů se vyskytuje na aktuálně vybrané virtuální ploše. Z toho je možné určit šířku vyhrazenou pro jeden titulek. Jestliže je počet klientů nenulový, vybarví se prostor pro první titulek zvýrazňující barvou, který představuje top-level okno s keyboard focusem. V prostoru mezi jednotlivými titulky se pro jejich oddělení vykreslí tenké svislé čáry.

V cyklu se procházejí všechny záznamy struktury `stack` a pro klienty, kteří patří do vybrané aktuální plochy, se zjistí titulek, který se následně vykreslí na spočtenou pozici. Pokud došlo k přijetí události typu `ButtonPress`, je zjištěné, jestli došlo ke kliknutí do okna pluginu. Jetliže ano, spočítá se z pozice, do kterého titulku se kliklo. Na klienta, se kterým tento titulek koresponduje, se volá funkce `raise_client` a `XRaiseWindow`, které způsobí, že se toto top-level okno vynoří na vrchní pozici.

plugin clock

Plugin clock slouží k vypisování aktuálního času s rozlišitelností minut. Aby se provedl kód pluginu clock, musí přijít událost typu `Expose` nebo událost `WM_TIMER`, která je generovaná správcem oken každou sekundu. Vytvoří se grafický kontext s patřičně nastavenými proměnnými a překreslí se okno pluginu. Postupným voláním funkcí `time` a `ctime` se získá textový řetězec obsahující časový údaj. Ten je posléze zkrácený, aby obsahoval čas ve formátu "hh:mm". Následně se voláním funkce `XChangeGC` změní proměnné grafického kontextu, aby bylo možné následně do okna pluginu viditelně vypsat získaný časový údaj.

4.8 Metriky kódu a ověřování funkčnosti

Počet souborů aplikace samotného správce oken je osm. Aplikace je dále rozšiřována pluginy. Celkem byly implemtovány čtyři pluginy. Každý plugin se sestává z jednoho souboru. Soubory aplikace mají 1450 řádků zdrojového kódu. Soubory čtyř uvedených pluginů mají dohromady 466 řádků zdrojového kódu. Velikost spustitelného souboru je 33484 bytů.

Ověřování funkčnosti správce oken probíhalo na operačním systému FreeBSD 6.2 a serveru Xorg 7.2.0 při rozlišení obrazovky 1024x768 pixelů. Testovací aplikace se spustily a bylo kontrolováné, jestli se chovají korektně. Názvy aplikací, pomocí kterých byl správce oken testováný, jsou uvedeny níže. K nim jsou dále uvedeny poznámky značící výsledky testů.

- xterm včetně textových aplikací (mc, vim, ...)
  - bez problemu
- xclock, xcalc, xedit, xpaint, xeyes, xdvi ...
  - bez problemu
- firefox 2.0.0.4, opera 9.24
  - bez problemu
- psi v0.10 jabber klient
  - bez problemu
- mplayer 1.0rc1
  - bez problemu
- gvim 7.1, gqview 2.0.4
  - bez problemu
- acroread 7.0.9
  - špatné umístění okna find do rámečku a nemožnost ho po uzavření opětovně otevřít
- gimp 2.2
  - z některých oken nelze vytvořit klienty
  - některá okna mají špatný rozměr rámečku

Kapitola 5

Závěr

Implementovaný správce oken se řadí do kategorie “reparenting window manager”. To znamená, že okna aplikace jsou orámována rámečkem s tlačítky. Ovládání aplikace se nese v duchu dnešní doby. Jsou podporovány virtuální plochy, panel pro spouštění, sledování a přepínání aplikací. Zrychlit práci umožňují klávesové zkratky. Funkcionalita správce oken je dále rozšiřovaná přídavnými pluginy, které jsou implementováné formou sdílených knihoven. Zvolené řešení implementace bylo provedeno s ohledem na minimální paměťovou náročnost a proto nebylo použito žádných vysokoúrovňových knihoven pro grafická uživatelská rozhraní. Zdrojový kód aplikace je dělen do šesti modulů. Každý modul je zaměřen na specifickou činnost správce oken.

Pro další pokračování vývoje správce oken by bylo vhodné zaměřit se na odstranění nedostatků zjištěných při testech 4.8. To znamená zajména nemožnost překonfigurování některých top-level oken aplikací nebo nemožnost z některých top-level oken aplikací vytvořit klienta. Bylo by vhodné zjistit příčinu tohoto chování a nedostatky odstranit. Je možné také dále rozšiřovat funkcionalitu správce oken např. zpracováváním dalších typů událostí. Podobně je možné více podporovat komunikaci pomocí ICCCM protokolu. Tato rozšíření ovšem nejsou nezbytně nutná. Správce oken totiž nemusí podporovat veškeré tyto prvky.

Přínosy projektu spočívají zejména ve vyzkoušení návrhu a implementace grafické aplikace v tak komplexním prostředí jako je X Window System. Správce oken je navíc vyjímečným případem, který musí podporovat speciální funkcionalitu, než je tomu u obyčejné grafické aplikace. Proto bylo nutné vyhledávat těžko dostupné informace, zejména pak v anglickém jazyce. Náročné se ukázalo být ladění takto nízkoúrovňově vyvíjené aplikace, kdy bylo nutné často přepínat do textového režimu. Při návrhu pomohlo i nahlížení do zdrojových kódů OpenSource projektu DWM [3].

Celá práce je koncipovaná tak, aby byly nejdříve vysvětlené obecné principy programování grafických aplikací v X Window Systému, později s důrazem na informace potřebné k vývoji správce oken. Proto předpokládám, že tato práce přispěje svou uceleností zejména k objasnění návrhu a vývoje správce oken.

Literatura

[1] Adrian Nye. *Xlib Programming Manual*. O'Reilly, 1990. ISBN 1-565920-02-3.

[2] Andrian Nye. *Xlib Reference Manual for Version 11*. O'Reilly, 1990. ISBN 0-937175-12-9.

[3] WWW stránky. Dynamic window manager. `http://www.suckless.org/wiki/dwm`.

[4] WWW stránky. Index xlib funkcí. `http://www.tronche.com/gui/x/xlib/function-index.html`.

[5] WWW stránky. Přehled správců oken. `http://www.xwinman.org/`.

[6] WWW stránky. Xlib programming manual. `http://www.sbin.org/doc/Xlib/`.

Dodatek A

Přílohy

A.1 Instalace správce oken

Zdrojové kódy správce oken se přeloží příkazem make z adresáře wm/. Zdrojové kódy pluginů lze přeložit příkazem make z adresáře wm/plugins. Do souboru .xinitrc je potom potřeba doplnit následující text, který nastaví barvu root okna, zajistí zasílání časových údajů a provede automatický start správce oken po příkazu `startx`:

```
xsetroot -solid gray     # nastaví barvu root okna

while sleep 1;           # každou sekundu pošle správci oken
    do date;             # časovou signalizaci
done | /cesta/wm/wm;

exec /cesta/wm/wm        # spustí správce oken
```

Pro běh pluginu je potřeba, aby byly sdílené knihovny pluginů umístěny v adresáři wm/plugins/. Konfigurace pluginů se provádí pomocí souboru wm/plugins/plugins.conf. V tomto soubou patří každému pluginu jeden řádek, který má následující význam:

```
název_souboru_pluginu   šířka_v_pnelu   0-vlevo/1-vpravo
```

A.2 Ovládání správce oken

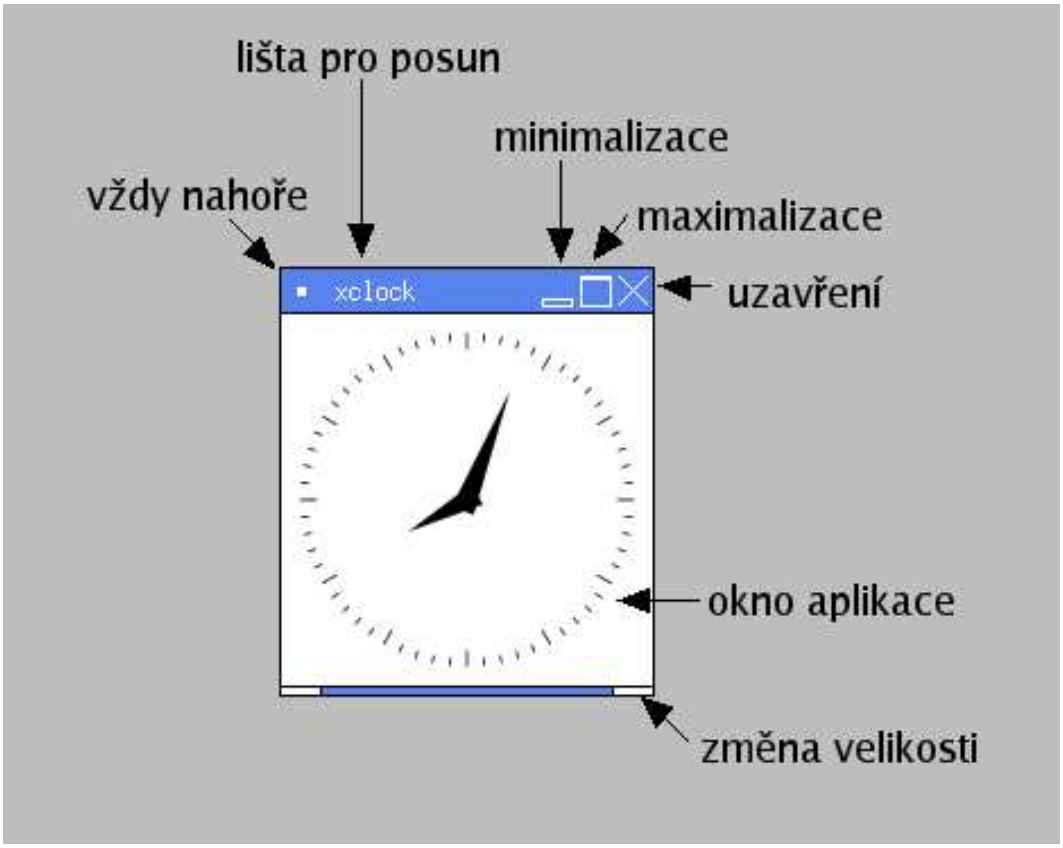
Klávesové zkratky

Při práci se správcem oken lze použít následujících klávesových zkratek:

```
přepínání virtuálních ploch        Alt + [F1 - F12]
přeposlání okna na jinou plochu    Alt + Shift + [F1 - F12]
cirkulace okny                     Alt + Tab
spuštění xtermu                    Alt + x
ukončení běhu správce oken         Alt + q
```

Manipulace s oknem

Pro manipulaci s oknem slouží tlačítka v jeho orámování. Jejich funkce jsou popsány na obrázku A.1.



Obrázek A.1: Použití tlačítek okna

Plugin menu

Pro vysunutí nabídky kategorií programů je nutné nejdříve kliknout na tlačítko `WM Menu`. Když je zobrazená nabídka kategorií programů, je možno pohybem kurzoru po titulcích kategorií zobrazovat nabídky programů. Po vybrání požadované kategorie programů se dá pohybem kurzoru z této nabídky přesunout do nabídky programů a kliknout na požadovaný titulek programu. Následně se obě nabídky uzavřou a dojde ke spuštění požadovaného programu. Uzavření nabídek lze případně dosáhnout i stiskem tlačítka mimo uvedené nabídky.

Plugin flats

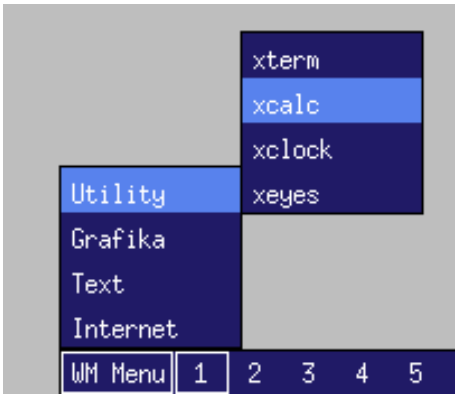
V grafice tohoto pluginu jsou zobrazená okénka symbolicky znázorňující všechny virtuální plochy. Pro přepnutí na danou virtuální plochu je nutné kliknout do odpovídajícího okénka. Barevně je znázorněné, na kterých virtuálních plochách se nachází alespoň jedno top-level okno a která virtuální plocha je aktualně nastavená.

Plugin box

Grafika pluginu box zobrazuje titulky oken aplikací v pořadí, v jakém jsou navzájem zanořena na obrazovce jim příslušná okna. Pro vynoření některého okna je nutné kliknout na odpovidající titulek. Titulek nejvýše umístěného okna je barevně zvýrazněný.

Plugin clock

Plugin clock zobrazuje aktuální čas ve formátu “hh:mm”. Žádná další interakce s uživatelem není podporována.



Obrázek A.2: Plugin menu

Obrázek A.3: Plugin flats a plugin clock

Obrázek A.4: Plugin box